UNIVERSIDADE FEDERAL DE SANTA CATARINA

YURI KAYSER DA ROSA
EVERTON SCHAFASCHEK COELHO

SPACE CODE: UM JOGO PARA O DESENVOLVIMENTO DE PENSAMENTO COMPUTACIONAL UTILIZANDO INTERFACES TANGÍVEIS

FLORANÓPOLIS-SC
2018

YURI KAYSER DA ROSA
EVERTON SCHAFASCHEK COELHO

SPACE CODE: UM JOGO PARA O DESENVOLVIMENTO DE PENSAMENTO COMPUTACIONAL UTILIZANDO INTERFACES TANGÍVEIS

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação, da Universidade Federal de Santa Catarina, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de Informação.

FLORANÓPOLIS-SC
2018

YURI KAYSER DA ROSA
EVERTON SCHAFASCHEK COELHO

SPACE CODE: UM JOGO PARA O DESENVOLVIMENTO DE PENSAMENTO COMPUTACIONAL UTILIZANDO INTERFACES TANGÍVEIS

Trabalho de Conclusão de Curso apresentado ao curso de Sistemas de Informação, da Universidade Federal de Santa Catarina, como requisito parcial para a Obtenção do grau de Bacharel em Sistemas de Informação.

FLORANÓPOLIS-SC, 29 de Novembro de 2018

BANCA EXAMINADORA

João Bosco Mangueira Sobral
Universidade Federal de Santa Catarina

Edla Maria Faust Ramos
Universidade Federal de Santa Catarina

José Eduardo De Lucca
Universidade Federal de Santa Catarina

Dedicamos este trabalho a todos os amigos e familiares que de alguma forma colaboraram e se esforçaram para que este trabalho rendesse os frutos desejados. Menção especial para Guilherme Kanarek, que nos ajudou a desenvolver a parte visual do jogo criado, e à Carol Prieto que nos ajudou na correção do texto aqui apresentado.

RESUMO

Desde que Wing (2006) cunhou o termo, o Pensamento Computacional vem sendo destacado por diversos autores (como BLIKSTEIN 2008 a STEPHENSON 2011) como uma nova e fundamental ferramenta para ler, compreender e dar soluções a problemas em um mundo cada vez mais digital.

Atento a importância deste conceito e com o objetivo de ajudar na difusão e aprendizado do mesmo, este trabalho apresenta a criação do Space Code, um jogo educacional na plataforma Android cujo objetivo é servir como uma ferramenta de apoio ao desenvolvimento de Pensamento Computacional em crianças entre 7 a 12 anos.

O jogo criado faz uso Interfaces Tangíveis que os usuários manipulam para montar algoritmos e assim resolverem os problemas a que são expostos. A aplicação desenvolvida também utiliza técnicas de Visão Computacional para identificar os objetos tangíveis e performar a solução proposta.

Palavras-chave: Pensamento Computacional, Interfaces Tangíveis, Jogos Educacionais, Visão Computacional, Jogo, Aplicativo, Mobile, Android.

ABSTRACT

Since Wing (2006) coined the term, Computational Thinking has been highlighted by many authors (like BLIKSTEIN 2008 and STEPHENSON 2016) as a new and fundamental tool to read, comprehend and solve problems in a increasingly digital world.

Knowing how important this concept is and with the objective of helping the propagation and learning of it, this work presents the creation of Space Code, an educational game for the Android platform which objective is to serve as a support toll in the development of Computational Thinking for children between 7 and 12 years old.

The developed game uses Tangible Interfaces that are manipulated by the users to build algorithms and solve the problems to which they are exposed. The application also uses Computer Vision techniques to identify the tangible objects and perform the proposed solution.

Keywords: Computational Thinking, Tangible Interfaces, Educational Games, Computer Vision, Game, App, Mobile, Android.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

TUI	Tangible User Interfaces

SUMÁRIO

1 **INTRODUÇÃO**

O desenvolvimento de computadores cada vez mais baratos e pequenos trouxe estas máquinas das fábricas e universidades para dentro de nossas casas e salas de aula, possibilitando o acesso a estes dispositivos por pessoas das mais variadas idades e classes sociais. Segundo a 28ª Pesquisa Anual de Administração e uso de Tecnologia da Informação nas Empresas, realizada pela Fundação Getúlio Vargas de São Paulo (FGV, 2017) , o número de *smartphones* no Brasil em Maio de 2017 era de 198 milhões, sendo maior que computadores, 166 milhões, o que resulta em uma densidade de 1,8 dispositivos por habitante (MEIRELLES, 2017).

Tal avanço permitiu que os conceitos de Ciência da Computação atingissem cada vez mais outras áreas do conhecimento, fazendo com que estas utilizem técnicas computacionais para resolver problemas complexos de suas funções. Dessa maneira, a computação deixa de ser uma área de conhecimento de nicho para tornar-se cada vez mais interdisciplinar, algo que pode ser visto tanto na academia quanto no mercado de trabalho. Como bem exemplificado por BLIKSTEIN (2008, p. 1):

> Um engenheiro industrial, ao tentar redesenhar a linha de produção não usa só papel e lápis - usa modelos computacionais. Um economista tentando fazer uma projeção de inflação não faz as contas na cabeça - usa, claro, modelos.

Este novo mundo em que vivemos exige um extenso conjunto de habilidades para o pleno exercício da cidadania, como a computação e o Pensamento Computacional, e é justo oferecer às nossas crianças a oportunidade de aprender tais conceitos, para que possam ser consumidores e criadores educados de novas tecnologias que podem melhorar a vida de todos (BLIKSTEIN , 2008, p. 1). e (VON WANGENHEIM; NUNES; DOS SANTOS, 2014).

Entretanto, a forma de ensino atual deste conceitos pode ser vista como problemática, pois percebe-se que o ensino de computação nas escolas está voltado principalmente para o uso das tecnologias e não para a compreensão e aplicação das mesmas. (VON WANGENHEIM; NUNES; DOS SANTOS, 2014).

Além disso, a inserção de dispositivos computacionais possui uma série de outras barreiras, como, por exemplo, a falta de treinamento de professores para utilizar e integrar estas novas tecnologias no currículo, falta de suporte técnico para eventuais problemas com os dispositivos e também a ausência de equipamentos suficientes para todos os alunos, fato que por vezes obriga os professores a organizar as atividades de tal forma que alguns alunos estão interagindo com os computadores e outros estão realizando outras tarefas (KLEIMAN, 2000).

Neste contexto, o tema Interfaces Tangíveis chama a atenção e apresenta

oportunidades de criação de sistemas físicos computacionalmente aumentados, indicando vantagens tanto no âmbito digital (como as possibilidades de edição e manipulação de forma diferenciada) quanto no fator físico, fazendo com que a interação entre homem e sistema computacional não esteja centrada apenas no dispositivo e sim no ambiente. Isso permite, por exemplo, que todos os alunos possuam os meios necessários para realizar suas atividades.

FALCÃO e GOMES (2007) sugere que o uso de Interfaces Tangíveis para a educação traz vantagens como maior engajamento sensorial (uso de mais sentidos para construir o conhecimento), acessibilidade, aprendizagem em grupo e aprendizagem divertida, o que estimula a comunicação, reflexão, imaginação, criatividade em diferentes níveis de abstração e aumenta a consciência sobre a experiência vivenciada.

Tendo esses pontos em vista, este projeto consiste na criação do jogo Space Code, um jogo em plataforma *mobile* que serve como apoio ao desenvolvimento de Pensamento Computacional em crianças na faixa etária de 7 a 12 anos, através da exposição à desafios que devem resolvidos aplicando conceitos relacionados ao tema, e que promovam a interação entre crianças, computadores e objetos físicos os quais estas devem manusear para para solucionar os problemas propostos.

1.1 OBJETIVO GERAL

Este trabalho tem como objetivo o desenvolvimento de um aplicativo que sirva como apoio para o desenvolvimento de Pensamento Computacional em crianças por meio de um jogo educativo, utilizando Interfaces Tangíveis como meio de interação entre usuários e aplicação.

1.2 OBJETIVOS ESPECÍFICOS

I. Analisar a fundamentação teórica referente à Pensamento Computacional e à Interfaces Tangíveis
II. Analisar o estado da arte de trabalhos e aplicações voltadas ao desenvolvimento de Pensamento Computacional em crianças;
III. Criação de um aplicativo *mobile* voltado para o desenvolvimento de Pensamento Computacional em crianças utilizando Interfaces Tangíveis.

1.3 ORGANIZAÇÃO DO TEXTO

Este trabalho é organizado em oito capítulos. O primeiro capítulo traz uma

introdução sobre o tema abordado no projeto, bem como os objetivos e a justificativa. No segundo capítulo é feita a fundamentação teórica de temas importantes para o entendimento do projeto, apresentando assuntos como Pensamento Computacional e Interfaces Tangíveis.

No terceiro capítulo é exposto o estado da arte relativo a aplicações e trabalhos voltados para o ensino Pensamento Computacional para crianças. O quarto capítulo trata da proposta do aplicativo a ser produzido. O quinto capítulo apresenta o jogo proposto, mostrando fluxos de navegação, regras e objetivos. No sexto capítulo é descrito o processo de desenvolvimento, mostrando como foram escolhidas as ferramentas utilizadas durante o processo e como algumas partes do código da aplicação funcionam. No sétimo capítulo são apresentados os resultados do trabalho realizado. Por fim, no oitavo capítulo são feitas as considerações finais sobre o projeto e sugestões para trabalhos futuros.

2 **FUNDAMENTAÇÃO TEÓRICA**

2.1 PENSAMENTO COMPUTACIONAL

Pensamento Computacional é um termo cunhado por Wing (2006) para uma forma de pensar e encontrar soluções para os problemas de diversas situações do dia a dia utilizando como base os fundamentos da Ciência da Computação e Matemática. Segundo Wing (2006). Pensamento Computacional não é saber programação e sim pensar como um cientista da computação. A pesquisadora ainda afirma que essa deveria ser uma habilidade básica, assim como ler, escrever, falar e fazer operações aritméticas.

Posteriormente, Wing (2014, p. 1) define o termo como o processo de pensamento envolvido em formular um problema e suas soluções de uma forma que um computador - homem ou máquina - consiga efetivamente implementar.

Complementando Wing, BLIKSTEIN (2008) afirma que Pensamento Computacional não é saber realizar as tarefas básicas no computador, como navegar na internet ou enviar um e-mail, mas sim utilizar o computador a fim de incrementar o poder cognitivo e operacional humano de realizar suas tarefas, aumentando a produtividade, inventividade e criatividade.

Porém, efetivamente, o que compõe o Pensamento Computacional? A Computer Science Teachers Association (CSTA) e International Society for Technology in Education (ISTE) listaram nove conceitos e capacidades para descrever o Pensamento Computacional (CSTA, 2011).

- **Levantamento de dados:** É o processo de encontrar e coletar dados apropriados para um problema;
- **Análise de dados:** É a capacidade de através de um conjunto de dados, encontrar sentido, padrões e tirar conclusões dos mesmos;
- **Representação de dados:** Utilizar de estruturas de dados, como gráficos, palavras, imagens, entre outras, para descrever e organizar os mesmos;
- **Decomposição de problemas:** Ao enfrentar um problema grande, quebrar em tarefas menores, facilitando sua resolução, tornando o problema gerenciável;
- **Abstração:** Criar abstrações para reduzir a complexidade de problemas enfrentados
- **Algoritmos:** Pensar e ordenar uma sequência de passos para resolver um problema ou alcançar algum objetivo;
- **Automação:** Fazer com que computadores ou máquinas realizem tarefas

tediosas ou repetitivas;

- **Simulação:** Representar e modelar processos, bem como experimentar os modelos criados;
- **Paralelização:** Identificar e organizar recursos que podem ser executados em paralelo para alcançar um objetivo em comum.

Para se entender como os conceitos acima podem ser utilizados no dia a dia, pode-se usar como exemplo um cientista que deseja realizar experimentos sobre um determinado tema.

Para realizar o estudo, o cientista teria que primeiramente criar abstrações sobre o problema proposto, com o objetivo de reduzir sua complexidade e decompô-lo em partes menores, o que o ajudaria a traçar as melhores estratégias para resolver cada parte do problema. Depois, ele poderia criar algoritmos de como realizar o experimento, simular, coletar os dados resultantes e analisá-los.

Existe um conjunto de habilidades ou pré-disposições que ajudam e aprimoram o aprendizado do Pensamento Computacional, como, por exemplo, a confiança em lidar com a complexidade dos problemas, a persistência em trabalhar com desafios, a capacidade de lidar com ambiguidade e problemas em aberto, bem como conseguir trabalhar com outras pessoas, conhecendo seus pontos fortes e fracos para alcançar um objetivo ou solução em comum (BARR; STEPHENSON, 2011).

2.2 INTERFACES TANGÍVEIS

Para compreender o que são Interfaces Tangíveis, é interessante definir o que, afinal, é uma interface. Interfaces podem ser entendidas como uma camada de comunicação entre dois elementos: um usuário que emite comandos e um artefato ou sistema que responde a esses comandos, promovendo assim uma interação (JETTER apud VIEIRA DOS REIS e GONÇALVES 2016).

As Interfaces Tangíveis podem ser definidas como o uso de objetos físicos, imbuídos de propriedades digitais, como sendo meios de entrada e/ou saída de uma aplicação, tirando proveito das vantagens da manipulação de objetos reais e das formas de interação providas pela tecnologia.

Fishkin (2004) exemplifica o funcionamento de uma TUI da seguinte forma:

- Um usuário interage com um objeto do dia a dia, geralmente movimentando, apertando ou manipulando o objeto com suas mãos. Tal interação gera um evento de entrada no sistema.
- A aplicação detecta esta interação e altera seu estado.
- A aplicação provê alguma forma de feedback para o usuário. Este evento

de saída se dá alterando a natureza física de algum objeto. Seja alterando o conteúdo exibido em sua tela, movimentando o objeto, alterando suas dimensões, etc.

Fishkin (2004) também propõe uma taxonomia para as Interfaces Tangíveis. Em seu artigo, o autor salienta dois pontos como agregadores de Interfaces Tangíveis. Um destes pontos remete ao tipo de metáfora a qual uma interface remete. No âmbito de TUI, este termo refere-se ao quão próximos estão da realidade os efeitos que a interação com uma interface tangível gera em um sistema. Com base nisso, foram criados cinco grupos de separação**:**

- **Nenhuma Metáfora:** Neste grupo, estão interfaces que não utilizam nenhuma metáfora, ou seja, o que acontece com a interface não possui uma conexão direta com o que acontece no mundo real. Como exemplo, podemos imaginar uma bola, que ao ser apertada altera um som emitido pelo sistema.
- **Metáfora de Nome:** Neste tipo de metáfora um objeto X no sistema é também um objeto X no mundo real, porém as ações realizadas nestes objetos não são análogas. Nestes casos, as similaridades ficam mais relativas aos aspectos físicos dos objetos, como forma, som ou cor. Como exemplo, podemos citar um sistema em que, ao colocar um carrinho sobre uma tela, uma representação digital de um carro é gerada no sistema.
- **Metáfora de Verbo:** Neste grupo, uma ação X no sistema é análoga a uma ação X no mundo real, independentemente do objeto em que isto é realizado. Como exemplo, podemos citar uma aplicação em que ao realizar um gesto de arremesso, um objeto é arremessado no sistema.
- **Metáfora de Nome e Verbo:** Aqui as analogias feitas dizem respeito tanto às características dos objetos quanto às ações. Neste tipo de interface, ao movimentar um objeto que representa um carro até o fim da tela encostando nas bordas, gera a representação no sistema de um carro em uma colisão.
- **Metáfora Completa:** Não são necessárias analogias entre nomes e ações do mundo real e suas representações no sistema, pois o mundo real e o mundo virtual são “iguais”. Um exemplo pode ser dado ao utilizar uma caneta stylus para escrever em um documento em um tablet, onde, quando o usuário escreve com a caneta sobre a tela, o sistema gera o que foi escrito no sistema.

O outro tipo de classificação feita por Fishkin (2004) está relacionada ao quão acoplados estão os dispositivos de entrada e saída. Neste sentido, o autor criou quatro grupos:

- **Totalmente Acoplados:** Neste caso, os dispositivos de entrada e saída do sistema são os mesmos.

- **Acoplamento Próximo:** Neste caso, os dispositivos de entrada e saída não são os mesmos, mas estão bastante próximos, muitas vezes interligados.
- **Acoplamento Ambiental:** Neste caso, o dispositivo de saída está no ambiente, nas periferias do dispositivo de entrada.
- **Acoplamento distante:** Neste caso, o dispositivo de saída está distante do dispositivo de entrada, muitas vezes até mesmo em outros ambientes.

2.2.1 **ENSINO COM INTERFACES TANGÍVEIS**

Entre os vários campos de aplicação das Interfaces Tangíveis, a área de ensino e aprendizado tem sido uma das que mais chama a atenção (MARSHALL, 2007). Isto se deve a uma visão geral do mundo educacional de que atividades práticas ou com uso de objetos manipuláveis facilitam o aprendizado. (MARSHALL, 2007).

Autores há muito tempo estudam as possíveis vantagens que interações físicas podem trazer ao aprendizado, como Piaget (1953), que demonstra, por exemplo, que crianças conseguem resolver melhor alguns problemas quando são apresentadas a algum tipo de material ou experiência física, do que quando tentam resolver o problema de forma simbólica. Por exemplo, crianças tendem a compreender melhor os conceitos de volume dos objetos quando podem interagir e realizar experimentos com os mesmos do que quando são apresentadas ao conceito simbólico de volume e aos números.

Sendo assim, segundo MARSHALL (2007) é possível que Interfaces Tangíveis proporcionem um aprendizado mais natural e efetivo, uma vez que frequentemente fazem uso de manipulações físicas concretas.

Outras vantagens decorrentes do uso de Interfaces Tangíveis podem ser a acessibilidade, fornecendo maiores opções de interação para crianças com necessidades especiais, e a aprendizagem em grupo, já que estas ferramentas podem ser criadas para estimular o trabalho em grupo,(Price et al. 2003 apud Falcão, 2007)

Utilizando as definições propostas por Mellar e Bliss (1994), Marshall (2007) sugere que as TUI podem ser de grande ajuda para a execução de dois tipos de atividades: Exploratórias e Expressivas.

No aprendizado através de atividades exploratorias, o aprendiz utiliza modelos já prontos para compreender novos conceitos. Neste tipo de atividade é necessário que o usuário compreenda e explore o modelo, incorporando os conhecimentos através de experimentações (MARSHALL, 2007; FALCÃO; GOMES, 2007).

MARSHALL (2007) propõe duas razões para as quais TUIs podem ser adequadas a este tipo de atividade. A primeira refere-se à capacidade das TUIs de promover interações mais fluídas e intuitivas e de proporcionarem um rápido feedback, o que facilita o processo de experimentação e teste. A segunda refere-se às teorias de que manipulações físicas de materiais podem ajudar no aprendizado. Assim, sendo as TUIs interfaces extremamente manipuláveis, elas podem proporcionar algum ganho de desempenho.

No aprendizado através de atividades expressivas, o aprendiz é instigado a criar suas próprias representações do domínio de conhecimento estudado. Ferramentas manipuláveis podem ajudar os estudantes a expressar suas ideias e a refletir sobre o modelo criado por eles.

MARSHALL (2007) propõe também duas justificativas para as quais TUIs podem ser adequadas a este tipo de atividade. A primeira refere-se à possibilidade de Interfaces Tangíveis registrarem as interações dos usuários de forma não explícita, permitindo aos alunos focar nas interações e posteriormente analisar os resultados. A segunda é que as TUIs são uma tecnologia inovadora e possuem potencial para criar modelos que podem não ser possíveis em mídias existentes atualmente.

FALCÃO e GOMES (2007) realizaram um estudo sobre as aplicações de Interfaces Tangíveis voltadas para a educação existentes na época, tendo analisado dois tipos de aplicações: aquelas que utilizavam TUI como dispositivos de entrada e Interfaces Gráficas como saída, e os chamados Manipulativos Digitais.

Na primeira categoria, os usuários podem interagir com objetos físicos e ver o resultado de suas interações na tela de um computador, já na segunda, os objetos tangíveis possuem tecnologia embarcada e são tanto o meio de entrada como o meio de saída.

Através da análise destas aplicações, os autores identificaram alguns conceitos que consideraram importantes para o desenvolvimento de aplicações tangíveis voltadas para a educação. Embora alguns destes conceitos sejam mais adequados a aplicação de Interfaces Tangíveis em sala de aula, outros, que são abaixo, podem abranger um contexto mais geral:

- **Acessibilidade:** O uso de recursos multimídia em artefatos tangíveis pode permitir que eles sejam adaptados para atender às necessidades especiais.
- **Independência do computador pessoal:** Principalmente nas aplicações do tipo Entrada TUI/Saída GUI há uma grande dependência dos computadores pessoais. É importante que as aplicações possuam maior independência, uma vez que, principalmente em escolas, pode haver uma certa dificuldade em inserir computadores pessoais no ambiente.

- ***scaffolding* e diferentes níveis de dificuldade:** É importante que as aplicações saibam lidar com diferentes níveis de dificuldade, fornecendo o que os autores chamam de *scaffolding*, que se refere às informações de ajuda que as aplicações fornecem aos usuários, tornando-os mais independentes da ajuda de um instrutor, ou fornecendo diferentes níveis de dificuldade dos exercícios, dando a opção ao usuário de escolher em que nível quer trabalhar.
- **Uso colaborativo:** A possibilidade de colaboração entre os usuários é um aspecto vantajoso. Objetos que permitam algum tipo de trabalho colaborativo podem aumentar o engajamento.
- **Engajamento do usuário:** Associações com o lúdico e a manipulação de objetos podem ser bastante atrativas aos usuários, gerando maior engajamento.
- **Simplicidade da interface:** As aplicações devem possuir interfaces simples, que não desenvolvam entraves ao processo de ensino-aprendizagem e que sejam de fácil compreensão.

As Interfaces Tangíveis podem também auxiliar no ensino de programação e Pensamento Computacional. Há uma grande variedade de aplicações classificadas como *Tangible Programing* que utilizam de interfaces totalmente tangíveis, ou que utilizam tanto interfaces gráficas como TUI para o ensino de computação, tendo como público alvo principalmente as crianças. Algumas destas aplicações podem ser vistas nos trabalhos correlatos citados no capítulo seguinte.

3 **TRABALHOS CORRELATOS**

Como trabalhos correlatos foram selecionados jogos voltados especificamente para o desenvolvimento de Pensamento Computacional. Os jogos foram separados em três grupos diferentes conforme suas características e os resultados estão descritos abaixo.

3.1 APLICAÇÕES DE INTERFACE PURAMENTE GRÁFICA

Nesta categoria estão agrupadas aplicações que propõem ensinar Pensamento Computacional para crianças com toda a interação do usuário ocorrendo através de interfaces puramente gráficas. Há ainda a subdivisão deste grupo em aplicações que utilizam Linguagens de Programação em blocos textuais, e aplicações que utilizam Instruções Simbólicas.

3.1.1 **LINGUAGENS DE PROGRAMAÇÃO EM BLOCOS TEXTUAIS**

Como exemplos desta categoria podemos citar o Scratch (MIT MEDIA LAB), no qual a criança pode utilizar uma linguagem de programação em blocos para criar sua estórias, jogos e animações, ou utilizar recursos criados por outros usuários.

Alguns jogos usam como base o Blockly (GOOGLE DEVELOPERS) , uma biblioteca que adiciona um editor de texto ao aplicativo desejado e representa conceitos de programação por meio de blocos interligados. Tal função permite que a estrutura criada gere um código sintaticamente correto em diferentes linguagens, como Java e Javascript.

Como ponto positivo deste grupo está o estímulo à criatividade, já que a possibilidade de criação de suas próprias animações pode trazer um maior engajamento da criança. O ponto negativo está na forma com que a criança insere os comandos, sendo completamente textual e menos intuitiva que as Interfaces Tangíveis ou em jogos que utilizam figuras para representar os comandos. Um exemplo é a aplicação Artista (CODE.ORG).

3.1.1.1 ARTISTA

O Artista é um jogo educacional criado pela Code.org, com o intuito de ensinar conceitos de Pensamento Computacional. O objetivo do jogo é, através de bloco de comandos, fazer o artista criar imagens e desenhos.

Para criar, a criança pode utilizar de comandos como avançar alguns pixels,

definir uma cor, mudar de direção, criar uma função e criar um laço de repetição. Ainda no jogo, é possível visualizar o código JavaScript correspondente aos blocos montados pela criança(CODE.ORG).

Figura 1 - Jogo Artista

Fonte: code.org

No próprio site da Code.org é possível encontrar muitos jogos parecidos com o mesmo objetivo, mudando os personagens, mas com o conceito de, através de blocos de comandos, fazer os personagens realizarem uma tarefa específica.

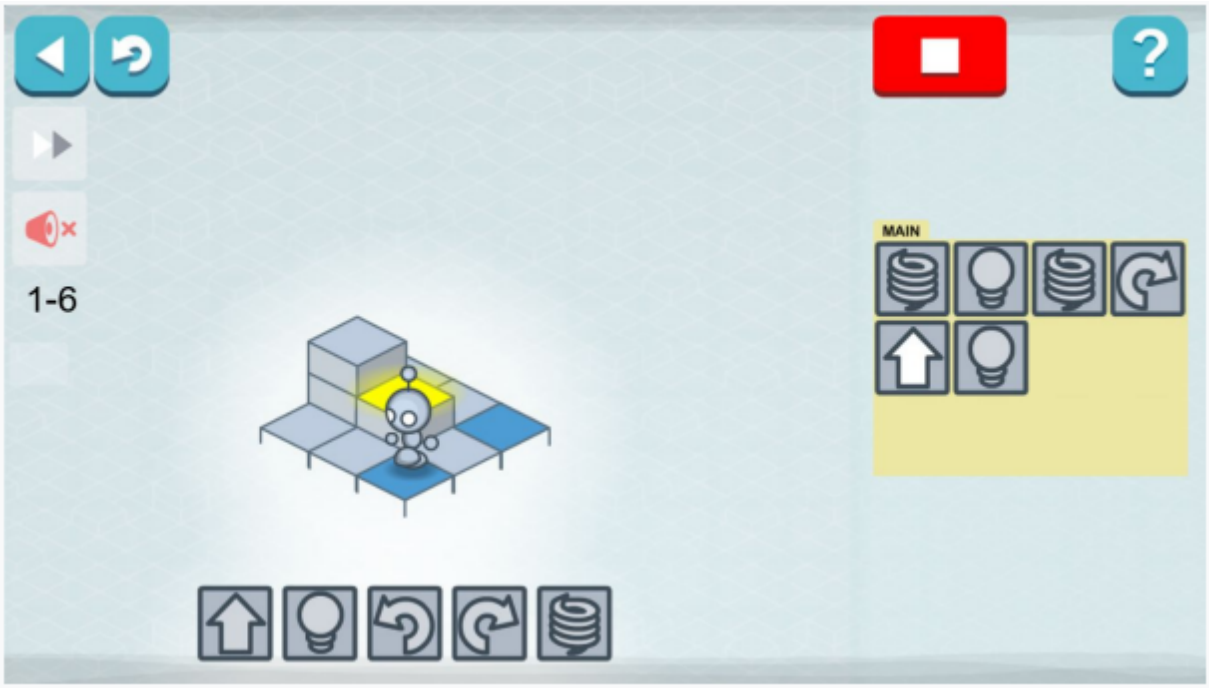
3.1.2 **JOGOS COM INSTRUÇÕES SIMBÓLICAS**

Neste grupo estão reunidas aplicações que utilizam símbolos como instruções as quais os usuários podem utilizar para montar algoritmos. Os jogos aqui descritos são de fácil acesso, com versões mobile e web, mas não contam com as vantagens de engajamento e ensino descritas no capítulo de Interfaces Tangíveis. Como exemplo, podemos citar o jogo LightBot (LIGHTBOT INC).

3.1.2.1 LIGHTBOT

É um jogo educacional do estilo *puzzle*, desenvolvido pela companhia Lightbot Inc. e projetado por Danny Yaroslavski, que tem como foco o ensino de algoritmos e lógica de programação. No jogo, o usuário deve organizar um conjunto de instruções com o objetivo de mover o Lightbot por uma plataforma e acender todos os quadrados azuis.

Figura 2 - Interface do jogo LightBot

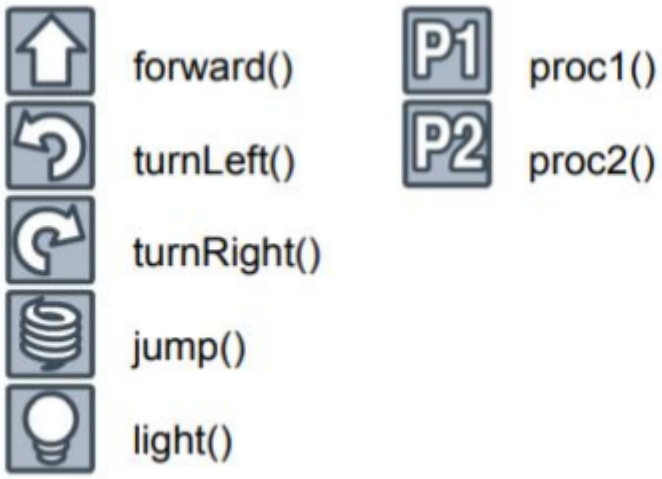


Fonte: LIGHTBOT INC

O jogo ensina práticas de programação como o planejamento, ao visualizar o problema da fase e projetar como resolvê-lo; algoritmos, indicando a sequência de instruções para resolver o problema proposto; teste, ao rodar as instruções e verificar se geraram os resultados esperados e debugging, caso não consiga resolver o problema, analisando e alterando a sequência de comandos para serem executados novamente (YAROSLAVSKI, 2014).

O jogo apresenta também conceitos de controle de fluxo, sequência de instruções, com a ordem correta dos passos, funções, ajudando no reconhecimento de padrões e reuso de comandos e repetições, sendo útil em tarefas repetitivas (YAROSLAVSKI, 2014).

Figura 3 - Lista de comandos do LightBot



Fonte: LIGHTBOT INC

Na figura 3, são listados os comandos disponíveis ao usuário no jogo para controlar o robô e cumprir o objetivo. Em cada fase o jogador recebe o desafio, planeja como resolver o problema, organiza os comandos necessários e executa o algoritmo proposto. É possível acompanhar o LightBot em cada passo da sequência, verificando, em caso de falha, qual momento ocorreu o problema (LIGHTBOT INC).

O jogo pode ser baixado gratuitamente, e possui a versão completa em mobile tanto para Android quanto IOS, possuindo também uma versão de demonstração criada para a Hour of Code que pode ser acessada via *Browser (LIGHTBOT INC).*

Outras aplicações seguem o mesmo estilo como o Robotizen (MAGE STUDIO - KID GAME) e o SpriteBox (LIGHTBOT INC).

3.2 APLICAÇÕES DE INTERFACE PURAMENTE TANGÍVEIS

Nesta categoria estão reunidas aplicações que se propõem ensinar Pensamento Computacional utilizando Manipulativos Digitais, trazendo uma experiência diferente das outras aplicações. Nestes exemplos, a criança não precisa de um celular ou computador para ter a experiência completa com o jogo interagindo apenas com as TUIs, além de permitir a criação de novos desafios, não deixando o usuário preso às fases fechadas e estimulando a criatividade.

Porém, este grupo apresenta um problema de acessibilidade, uma vez que é preciso obter as peças para poder realizar as atividades. Um exemplo deste tipo de aplicação é o Robot Mouse (LEARNING RESOURCES).

3.2.1 **ROBOT MOUSE**

É um jogo educacional criado pela Learning Resources para ensinar conceitos relacionados ao Pensamento Computacional e codificação, como solução de problemas, pensamento crítico e analítico, lógica, entre outros (LEARNING RESOURCES).

Figura 4 - Jogo Robot Mouse

Fonte: Learning Resources

O objetivo do jogo é fazer com que o rato atravesse o labirinto e alcance o queijo como pode ser visto na Figura 04. O próprio jogador pode criar o desafio, não possuindo fases pré-determinadas como em outras aplicações. Para isso, utiliza-se das peças grandes e quadradas para montar o tabuleiro, e das peças de paredes para delimitar o caminho e criar túneis.

O kit contém também cartas de ações, que representam os movimentos para frente, para trás e rotações, mas elas servem apenas de apoio à solução, uma vez que, após encontrar a sequência de instruções para resolver o problema, o usuário deve pressionar os botões contidos no rato para definir a sequência a ser executada

(LEARNING RESOURCES).

Os pontos negativos do jogo ficam por conta da não gratuidade, podendo limitar o acesso e na simplicidade dos comandos em comparação a outras aplicações, não abordando o conceito de funções e repetições, por exemplo.

Também é possível citar o jogo Cubetto (PRIMO TOYS) , no qual o usuário utiliza um painel de controle para definir as instruções e mover um robô de madeira. O brinquedo Think & Learn Code-a-pillar (MATTEL) é outro exemplo, em que a criança pode controlar os movimentos e sons de uma lagarta através de segmentos conectados em sequência, montando o corpo do brinquedo que gera os movimentos desejados.

Figura 5 - Painel de controle e robô do jogo Cubetto

Fonte: PRIMO TOYS

3.3 APLICAÇÕES DE INTERFACE HÍBRIDA

Nesta categoria estão reunidas aplicações que se propõem ensinar Pensamento Computacional através de interfaces híbridas que utilizam Interfaces Tangíveis como meio de entrada e Interfaces gráficas como meio de saída. Como exemplo podemos citar o jogo Coding Awbie (OSMO).

3.3.1 **CODING AWBIE GAME**

É um jogo educacional, criado e disponibilizado pela Osmo, com o intuito de ensinar Pensamento Computacional para crianças de 6 a 12 anos, disponível para iPad e iPhone. O objetivo do jogo é controlar o personagem *Awbie* através de comandos para coletar os morangos espalhados pelo mapa (OSMO).

Figura 6 - Jogo Coding Awbie

Fonte: Osmo

Para realizar as ações, as crianças ligam os blocos físicos mostrados na Figura 06 que representam os comandos de andar, pular, pegar, repetição, sequência, dormir e multiplicadores, criando assim uma sequência lógica de passos

a serem executados pelo personagem no jogo.

Os blocos são conectados magneticamente e ligados a um bloco principal que representa o comando Executar e possui um botão que, quando pressionado, abre duas pequenas portas no topo do bloco e indicam ao Ipad, que “lê”os blocos a partir de um dispositivo acoplado na câmera, que a sequência deve ser executada (MOYNIHAN, 2016). O jogo se propõem a ensinar fundamentos de código, reconhecimento de padrões, sequenciamento, repetições e criatividade.

Figura 7 - Blocos com os comandos

Fonte: Osmo

Como ponto negativo está o fato de ser um jogo pago e exclusivo apenas para iPad e iPhone, além de ser necessário comprar o Osmo Base, um kit que contém a base e o dispositivo para ser acoplado à câmera. Como pontos positivos, o jogo apresenta uma interface simples e intuitiva, explora bem os conceitos de Pensamento Computacional como condições lógicas, repetições, passagem de parâmetros e é bastante interativo.

A Osmo também possui um outro que segue o mesmo estilo chamado Coding Jam (OSMO). Voltado para crianças de 7 a 12 anos, o objetivo é criar uma sequência de passos que resultarão em ritmos, melodias e harmonias. Outro exemplo deste grupo é o Cubico Games (PDM INC) , em que o usuário coloca os blocos de comandos em um painel que serão interpretados pelo aplicativo e guiarão o personagem pelo mapa.

4 **PROPOSTA**

Considerando a importância dos conceitos de Pensamento Computacional em uma sociedade cada vez mais tecnológica, como descrito na seção 2.1 deste trabalho, este trabalho propõe a criação de um aplicativo *mobile* no estilo Jogo Educacional para auxiliar no ensino de conceitos como algoritmos, levantamento de dados, decomposição de problemas e abstração.

Levando em consideração as vantagens de aprendizado proporcionadas pelas Interfaces Tangíveis elucidadas na seção 2.2, este projeto tem como proposta a utilização de uma abordagem de Interação Híbrida, na qual os usuários poderão manipular objetos tangíveis e o resultado de suas interações será exposto em uma interface gráfica contida no dispositivo *mobile*.

A partir da análise dos trabalhos correlatos citados no capítulo anterior, optou-se pela utilização de uma abordagem de Interação Híbrida, como visto nos trabalhos listados na seção 3.3, por proporcionar maiores possibilidades de engajamento que as ferramentas que utilizam apenas de interfaces gráficas.

Além disso, a Interação Híbrida também fornece uma alternativa mais barata e acessível em relação às aplicações puramente tangíveis, que muitas vezes são compostas por objetos munidos de computação embarcada.

A maior flexibilidade das Interfaces Híbridas também foi levada em consideração, uma vez que permitem a utilização de objetos que podem muitas vezes serem representados por simples pedaços de papel ou madeira.

Dessa forma, a aplicação funciona da seguinte maneira: os usuários serão expostos a um desafio, mostrado na interface gráfica, e utilizarão os blocos físicos para montar uma sequência de ações a serem realizadas para resolver o problema apresentado. Após a montagem da sua resposta, o usuário tira uma foto da solução, vendo os comandos serem executados graficamente.

Entre os trabalhos relacionados, o que mais se assemelha à aplicação proposta é o Coding Awbie Game, visto na seção 3.3.1. Ao avaliar esse projeto, entretanto, foram identificados alguns problemas que podem comprometer a experiência do usuário, como a necessidade de adquirir um *kit* que contém o equipamento necessário para a leitura do conjunto de instruções proposto, a aquisição do conjunto de peças para a montagem do mesmo, a falta de mobilidade do dispositivo, que fica preso a uma base fixa, limitando o espaço de interação, e a restrição de plataforma IOS, que possui dispositivos mais caros em relação aos que utilizam a plataforma Android e atingem uma quantidade de usuários menor, uma vez que mais de 80% de *smartphones* vendidos no mundo são da plataforma Android (IDC, 2016).

Como alternativa esta proposta apresenta um jogo gratuito, desenvolvido para a plataforma Android, visando uma maior abrangência, e com recursos de internacionalização, podendo ser utilizado por usuários de outros países e assim aumentando o alcance do jogo. Para facilitar o acesso às peças do jogo, são disponibilizadas especificações necessárias para que elas possam ser fabricadas por qualquer um utilizando o material que desejar como, por exemplo, papel, madeira ou similares.

Também foi elaborada uma maneira de conferir maior mobilidade ao dispositivo de leitura das instruções, uma vez que o mesmo não precisa ficar em uma posição fixa, o que pode aumentar a área útil de montagem das instruções.

5 **SPACE CODE**

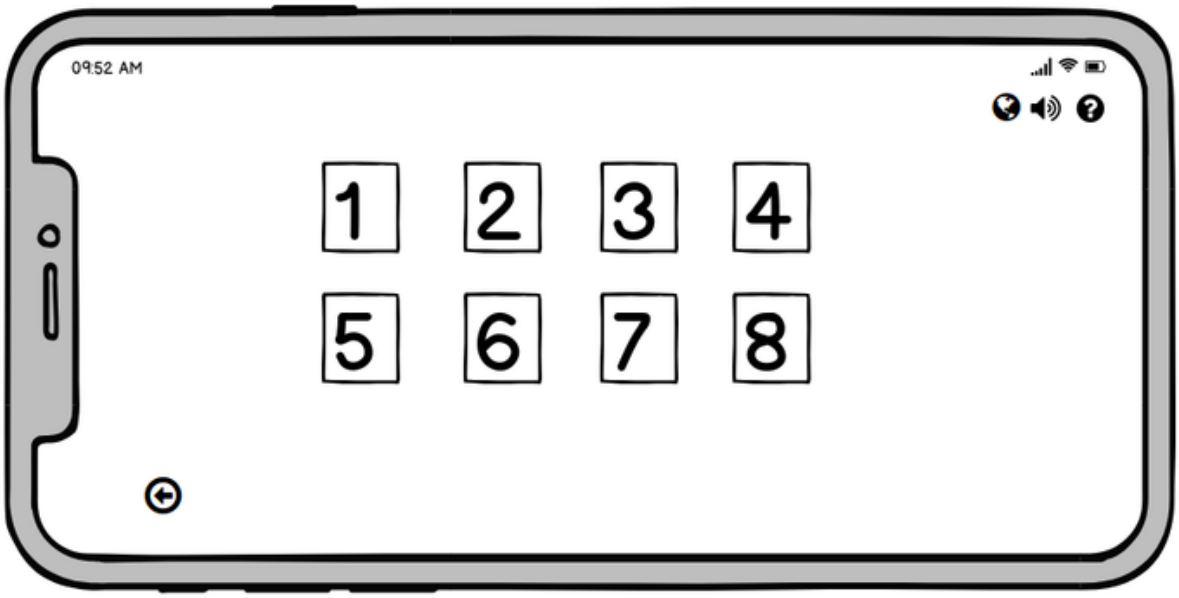
O jogo começa com uma tela simples que contém um botão do ícone de “*Play*” e botões com os ícones de “Ajuda”, “Som”, “Configuração de idioma” e "Download" no canto superior direito. Ao tocar no botão de “Ajuda” nesta tela, uma janela de diálogo é exibida contendo um texto explicativo sobre o que é o jogo e qual o seu objetivo.

Figura 8 - Tela inicial

Fonte: Os autores (2018)

Tocando no botão “Iniciar”, o usuário é guiado então para a próxima tela, onde são mostradas as fases disponíveis para jogar, representadas na figura 9 pelos botões numerados. O usuário também pode selecionar o botão de “Voltar”, para retornar à tela inicial do jogo.

Figura 9 - Tela de seleção de fases



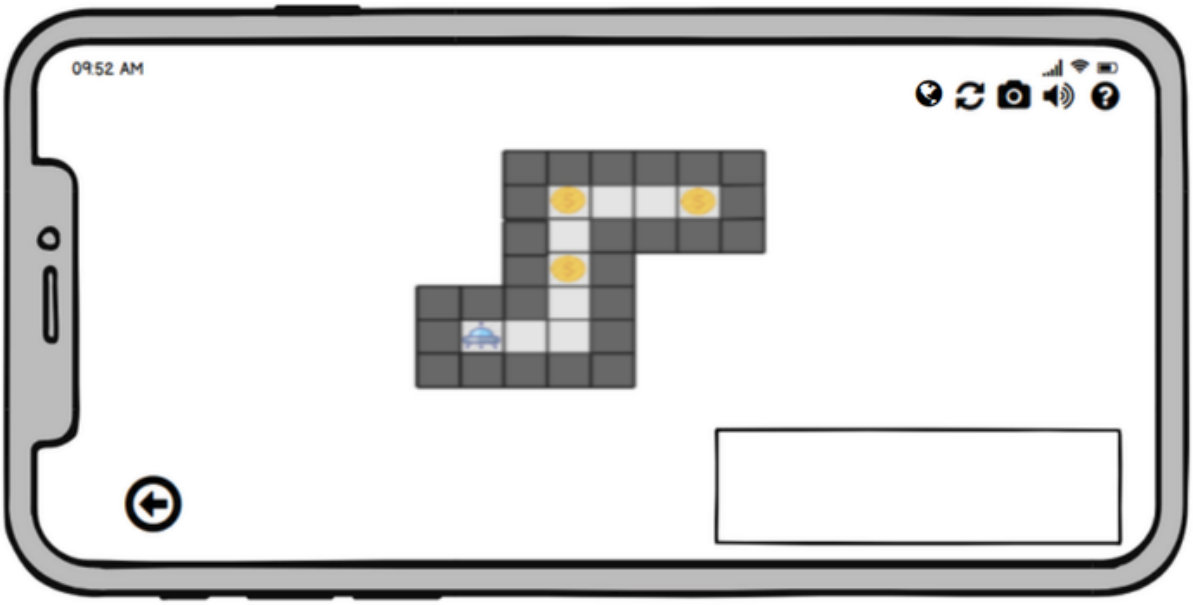
Fonte: Os autores (2018)

Ao tocar em algum dos botões numerados, o usuário é encaminhado para a tela da fase selecionada. São apresentados então o Tabuleiro disposto de acordo com o desafio especificado.

O objetivo do jogo é mover um foguete pelo tabuleiro com o propósito de pegar todas as estrelas. Para coletá-las, o usuário deve fazer com que a foguete passe pela mesma posição em que elas estão. Caso o foguete toque nas paredes ou em algum dos obstáculos o usuário perde.

Além de um tabuleiro, esta tela apresenta um botão com o ícone de “Câmera” na parte superior direita, junto aos botões de “Som”,“Configuração de idioma”, “Ajuda” e ”Recarregar Fase”, e um painel destinado a mostrar os comandos identificados após o processamento da foto capturada. Há também um botão simbolizando a ação de “Voltar” no canto inferior esquerdo da tela que, se acionado, retorna para a tela de seleção das fases.

Figura 10 - Tabuleiro



Fonte: Os autores (2018)

Caso toque no botão de “Ajuda” novamente será mostrada uma janela de diálogo, desta vez contendo um texto explicativo sobre os comandos que podem ser utilizados.

Figura 11 - Ajuda

Fonte: Os autores (2018)

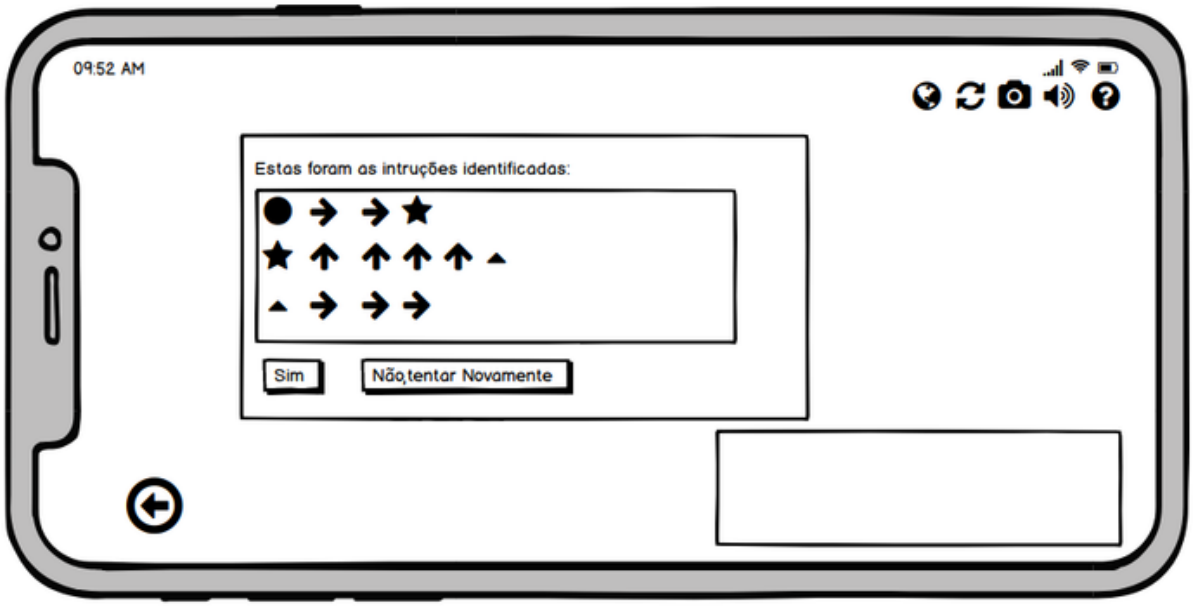
Ao acionar o botão “Câmera” o usuário será enviado para o aplicativo de

câmera do seu dispositivo para que possa fotografar as peças.

Após fotografar, deve ocorrer o processo de reconhecimento das peças na imagem. Caso haja algum problema durante essa etapa, ou seja, identificado que o algoritmo reconhecido não obedece às regras da linguagem, o sistema mostra um modal com mensagens de erro.

Após o reconhecimento das peças, o usuário é trazido de volta para o jogo e um novo painel aparece em primeiro plano mostrando as instruções identificadas, apresentando uma mensagem de texto perguntando se as instruções reconhecidas estão corretas e contendo os botões “Sim” ou “Não, tentar novamente”

Figura 12 - Painel com as instruções reconhecidas



Fonte: Os autores (2018)

Ao selecionar a opção “Sim”, o painel de confirmação dos comandos é fechado e o painel de comandos é preenchido com as instruções reconhecidas. Assim, o processamento das instruções é iniciado e o foguete move-se pelo tabuleiro. O sistema deve realizar a ação esperada para o comando, ao mesmo tempo em que sinaliza no painel de comandos qual instrução está sendo executada, e aguardar um segundo para a execução da próxima, tornando mais clara a percepção da ação para o usuário.

Caso o usuário selecione a opção “Não, tentar novamente” o aplicativo de câmera será aberto para que o usuário possa capturar uma nova foto dos comandos.

Após a execução de todos os comandos, é aberto um painel de encerramento da fase, informando se o usuário conseguiu concluir com sucesso ou não o desafio.

Também são exibidas as opções de “Selecionar fase”, que retorna à tela de escolha de fase, “Tentar novamente” que recarrega a fase atual e, em caso de vitória, a opção “Próxima fase”, que carrega o desafio seguinte para o jogador.

Caso a fase completada seja a última disponível para jogar, uma mensagem é apresentada ao usuário informando que ele completou todas as fases.

5.1 REGRAS DA LINGUAGEM

Todas as linguagens de programação possuem um conjunto regras as quais o usuário é obrigado a seguir para programar na mesma. Para este trabalho foram definidas algumas regras simples para que o algoritmo criado pelos usuários seja considerado válido. Procurou-se, criar uma linguagem simples, tendo em vista que o público alvo é de crianças.

Primeiramente, o código criado deve ter no máximo três linhas, sendo que o primeiro comando de cada linha precisa ser, obrigatoriamente, um comando de função: Estrela, Círculo ou Triângulo. As demais instruções inseridas na linha, são os comandos que serão executados pela função. Uma função pode ser definida em apenas uma das linhas. Ou seja, não é possível ter duas linhas que começam com uma mesma peça de função

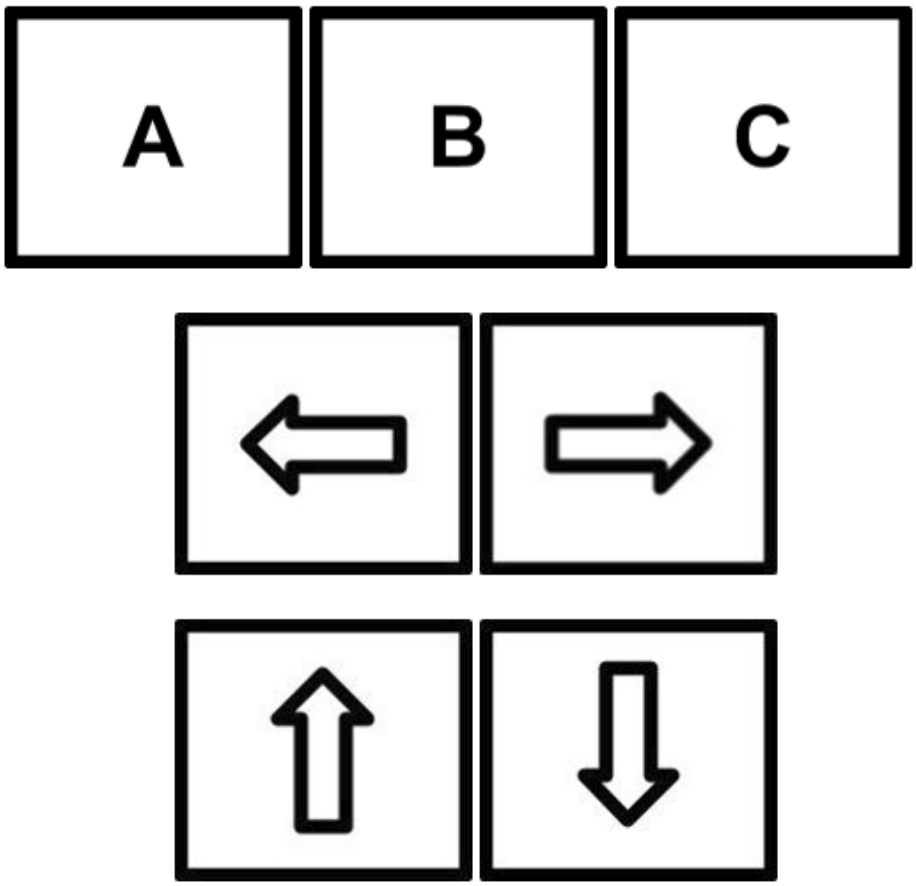
O código começa pela primeira linha de comandos. Para que alguma outra linha de comandos seja executada, a função que a representa deve ser inserida na linha. Se uma função é chamada, mas não está representada em nenhuma linha, o código é considerado inválido.

Caso haja um peça de *Loop* na linha, é necessário que ela seja seguida por pelo menos um comando, que não seja um *Loop* ou número, seguido de um número, que indica quantas vezes as instruções do *Loop* serão repetidas. Criação de repetições que não possuem números até o fim da função são consideradas inválidas. Números que não estão em uma estrutura de repetição, também são considerados como instruções inválidas.

5.2 PEÇAS

Ao longo do processo de desenvolvimento, foram testados dois modelos de peças. A primeira opção apresentada na figura 13, consistia de três peças que representavam as funções A, B e C, e por quatro peças direcionais que representavam a movimentação para cada uma das direções.

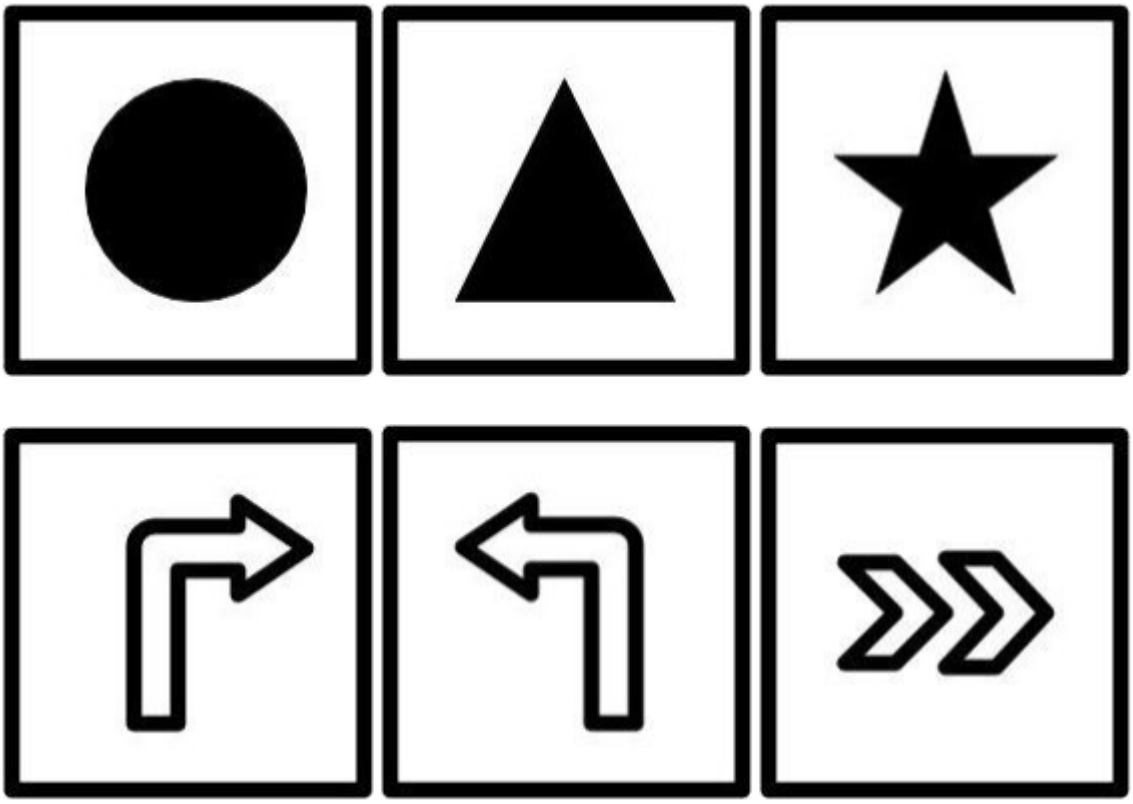
Figura 13 - Peças



Fonte: Os autores (2018)

Entretanto, durante o processo de desenvolvimento e testes, foi verificado que as peças com letras não possuíam bons índices de reconhecimento pelos mecanismos de Visão computacional, e também limitavam a criação de fases mais complexas principalmente quando havia necessidade de realizar movimentos repetitivos. Foi então proposto um segundo modelo de peças com a inclusão de estruturas de repetição, funções representadas por símbolos mais distinguíveis pelos mecanismos de Visão Computacional, e pela divisão da ação de movimento em duas peças diferentes, que representam a ação de indicar a direção para a qual o foguete deve rotacionar, e uma peça que indica o movimento na direção em que o foguete está dirigido.

Figura 14 - Funções e peças de movimentação

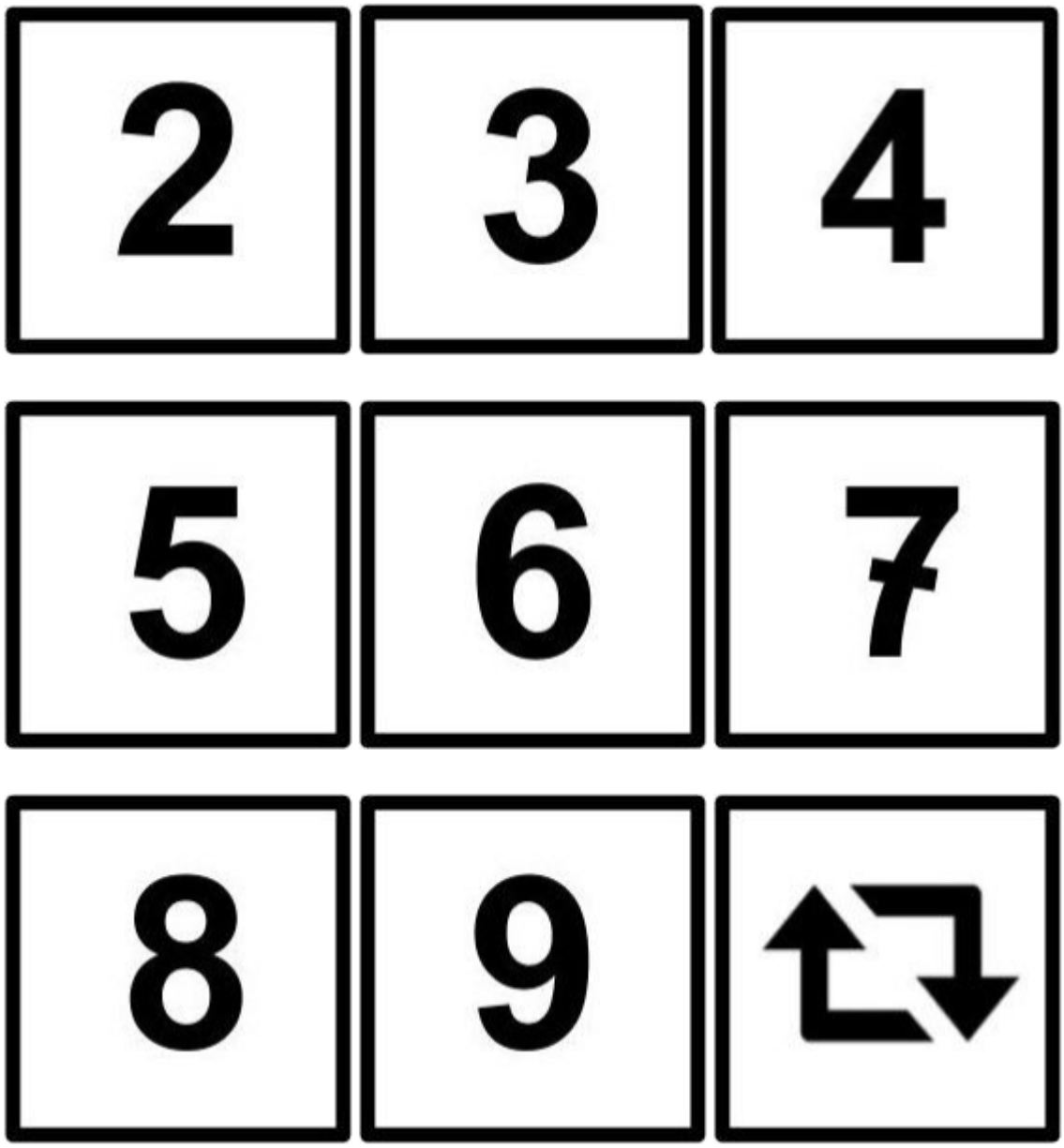


Fonte: Os autores (2018)

As peças de direcionamento, simbolizadas por setas direcionadas para a esquerda e para a direita, giram o foguete na direção indicada, enquanto a peça de movimento, simbolizada por duas consecutivas, move o foguete.

Também foram introduzidas peças que são utilizadas para criar estruturas de repetição. Estas consistem em peças de números de 2 à 9 e em uma peça que simboliza o Loop.

Figura 15 - Peças de repetição



Fonte: Os autores (2018)

Os objetos podem ser feitos de papel ou de qualquer outro material que o usuário desejar, mas devem seguir fielmente o modelo das figuras 14 e 15 com um fundo branco, bordas pretas e dimensões de 5cm x 5cm.

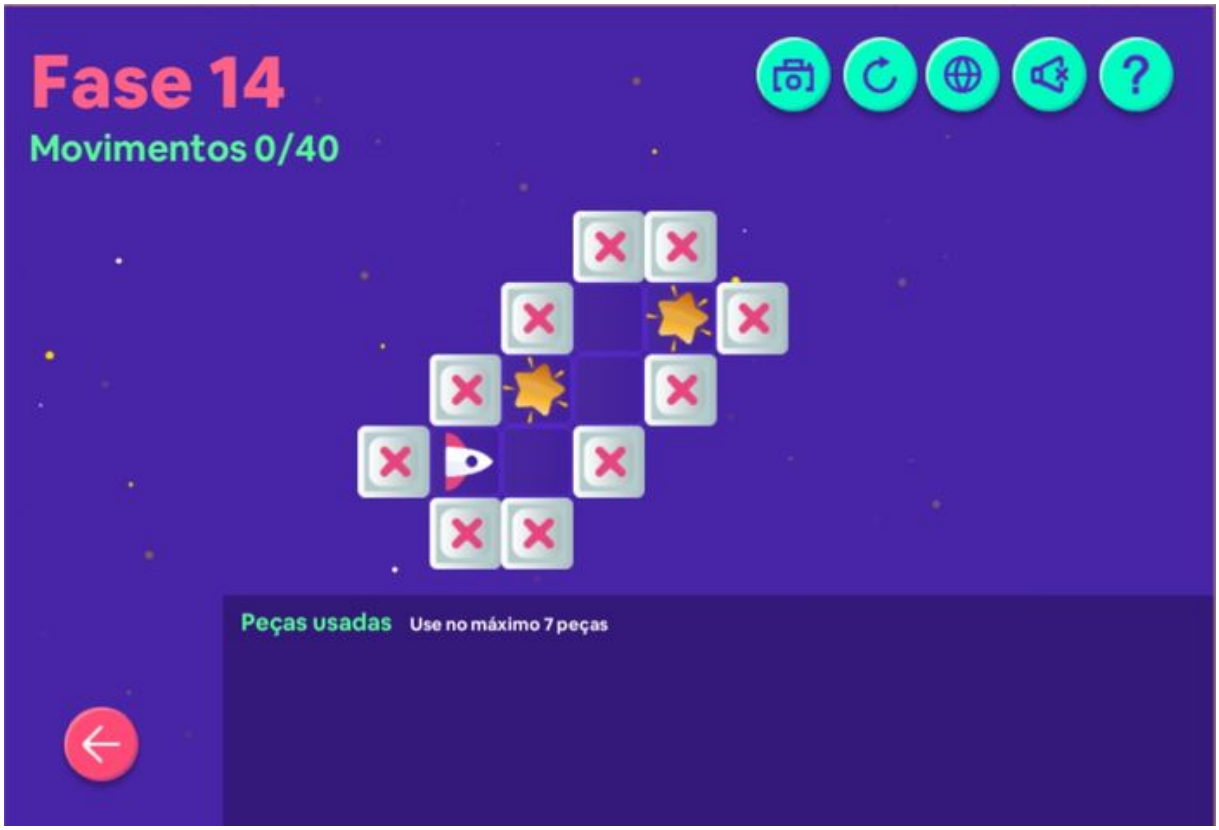
As peças são os elementos tangíveis do jogo e podem ser classificadas, de acordo com os conceitos listados nas seção 2.2, como de acoplamento próximo (geram efeitos em um dispositivo que está próximo) e metáfora de nome (a peça que representa girar para a direita, para o sistema também é girar para a direita).

5.3 FASES

Foram elaboradas 20 fases que inserem, gradativamente, todas as instruções e mecânicas do jogo. Os primeiros 4 níveis servem para introduzir o usuário às instruções de movimentação do jogo. Nos níveis seguintes são inseridos novos elementos, como alguns obstáculos que impedem a passagem do usuário, sendo necessário desviar para chegar até os coletáveis, e mais de um coletável por nível.

Conforme o usuário progride no jogo ele é também exposto a desafios cujas soluções são muito extensas, incentivando a utilização de funções, ou que exigem movimentos repetitivos e que poderão ser resolvidos através do uso das peças de repetição ou de recursão. Uma destas fases pode ser vista na figura 16.

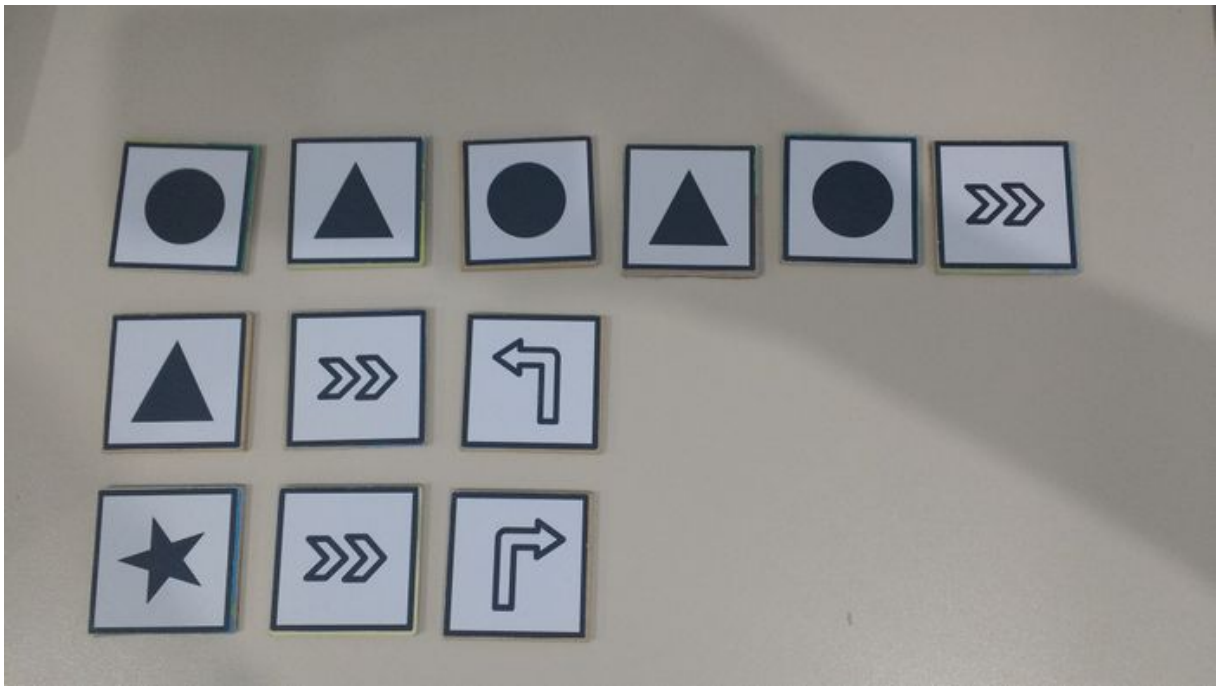
Figura 16 - Exemplo de fase



Fonte: Os autores (2018)

Esse nível possui elementos de parede, limitando a área em que a nave pode se movimentar, e dois coletáveis que podem ser capturados com um mesmo conjunto de movimentos. A figura 17 contém um exemplo de solução para este nível utilizando chamadas de funções.

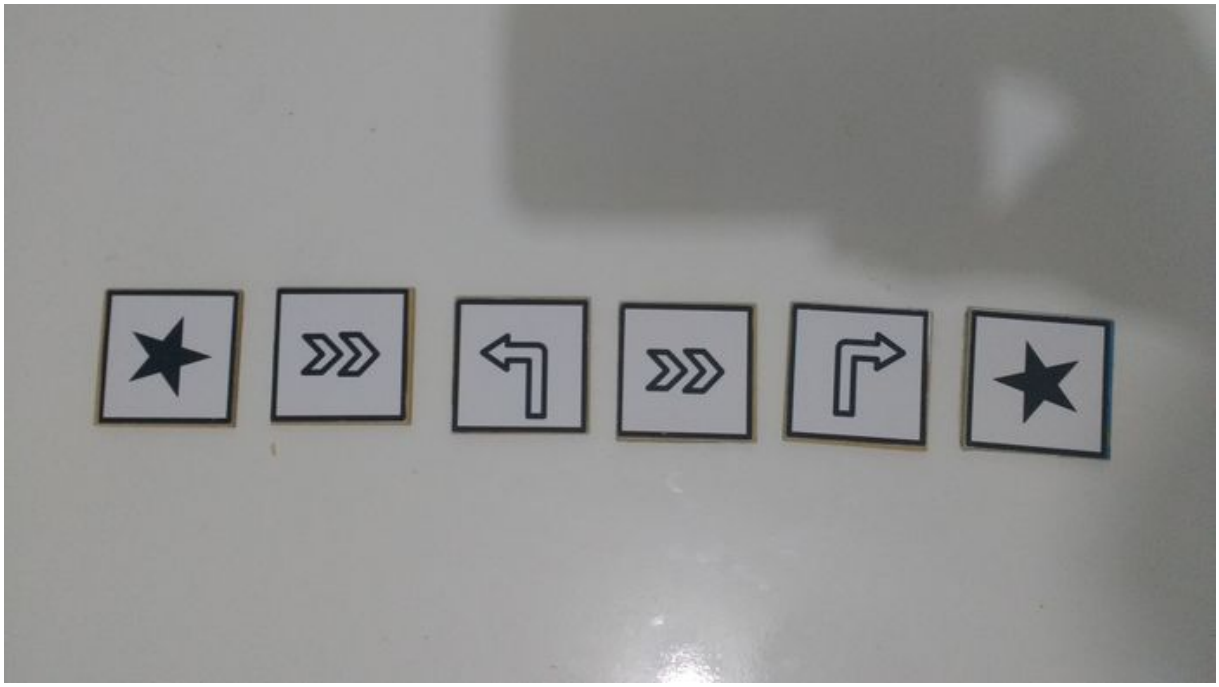
Figura 17 - Solução utilizando funções



Fonte: Os autores (2018)

A figura 18 apresenta uma solução para o mesmo problema, através do uso de recursão.

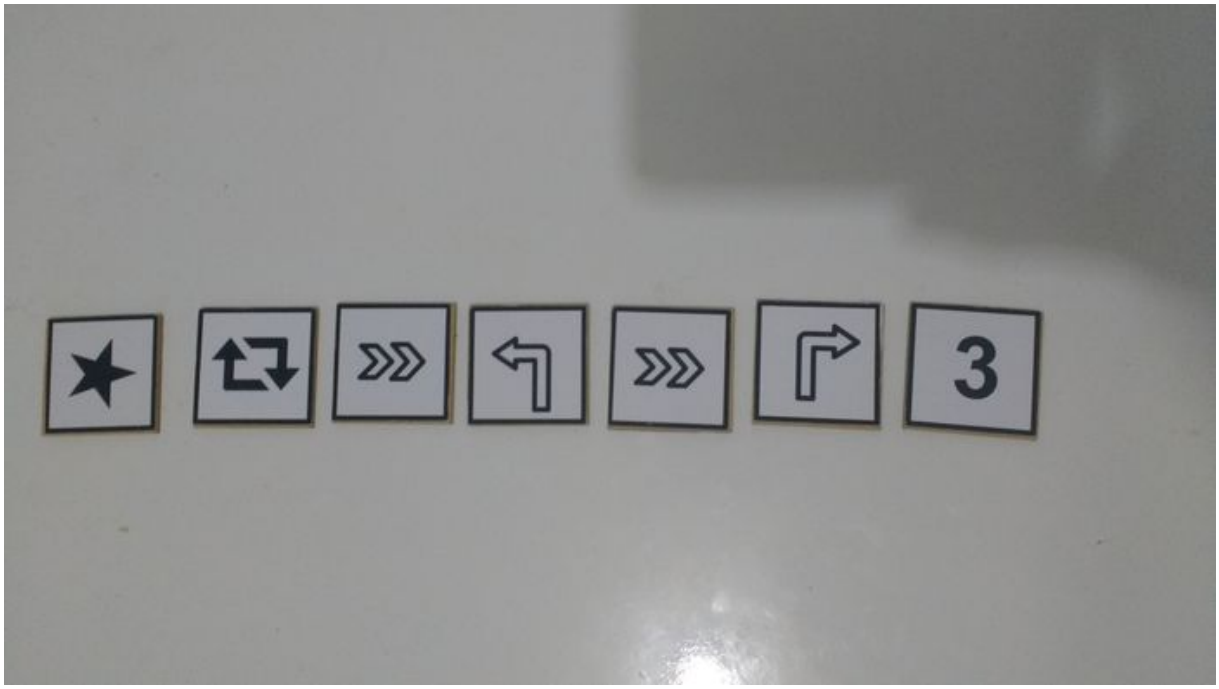
Figura 18 - Solução com recurção



Fonte: Os autores (2018)

A imagem 19 apresenta uma solução para o problema com o uso de Loop.

Figura 19 - Solução com Loop



Fonte: Os autores (2018)

Conforme o jogo avança novas mensagens de ajuda são apresentadas ao usuário, dando dicas sobre como funciona uma nova mecânica do jogo ou como utilizar determinadas peças.

Durante o processo de criação das fases, buscou-se identificar quais dos conceitos definidos pela (CSTA, 2011) como pertencentes ao escopo de Pensamento Computacional poderiam ser abordados por este trabalho.

- **Levantamento de dados:** Para resolver as fases o usuário deve identificar corretamente quais as instruções deve utilizar para montar o algoritmo.
- **Decomposição de problemas:** Para resolver algumas das fases mais complexas o usuário precisará utilizar funções e estruturas de repetição, particionando em cada uma delas um pedaço da solução do problema.
- **Algoritmos:** É o principal conceito desenvolvido pelo jogo. Para resolver os problemas a que é exposto o usuário deve montar, utilizando as peças, uma sequência de instruções que ao ser executada levará ao sucesso.
- **Simulação:** É um conceito representado pelo jogo pois o usuário pode ver o algoritmo criado utilizando as peças sendo executado no aplicativo, simulando as instruções definidas. 4

6 **PROCESSO DE DESENVOLVIMENTO**

Nesta seção é descrito como se deu o processo de desenvolvimento do software e dos demais artefatos de apoio ao seu funcionamento, abordando as fases de levantamento de requisitos, levantamento e definição das ferramentas utilizadas e elaboração de código.

6.1 REQUISITOS FUNCIONAIS

Contam como requisitos funcionais deste projeto as seguintes características:

- **RF-01:** Deve conter uma opção que permita ativar e desativar o som do jogo.
- **RF-02:** Deve conter um botão que, ao ser clicado, exibe mensagens de ajuda sobre o jogo.
- **RF-03:** Deve conter um botão que, ao ser clicado, mostra as opções de idioma do jogo, e ao selecionar uma opção, altera os textos do aplicativo para o idioma selecionado.
- **RF-04:** Deve permitir a seleção da fase que o usuário planeja jogar.
- **RF-05:** Deve exibir a quantidade limite de movimentos restantes da fase.
- **RF-06:** Deve exibir a quantidade máxima de peças que podem compor o algoritmo.
- **RF-07:** Deve permitir que o usuário, através da câmera do celular, capture fotos das peças que contêm as instruções para a solução dos desafios das fases.
- **RF-08:** Deve ser capaz de analisar a foto batida pelo usuário e identificar nela o algoritmo descrito pelo usuário através de técnicas de visão computacional.
- **RF-09:** Deve permitir que o usuário confirme que houve identificação correta das instruções, para o caso de elas não estarem de acordo com a disposição das peças.
- **RF-10:** Deve executar as ações vinculadas às instruções reconhecidas.
- **RF-11:** Deve mostrar em tempo real quais instruções estão sendo executadas.
- **RF-12:** Deve mostrar, ao finalizar a execução das instruções ou atingir o limite de instruções executáveis, uma mensagem informando se o jogador superou ou não o desafio.
- **RF-13:** Deve ser capaz de montar as fases do jogo baseadas em arquivos no formato JSON.

- **RF-14:** Deve permitir que o usuário reinicie a fase a qualquer momento.
- **RF-15:** Deve permitir que o usuário faça download de um modelo das peças utilizadas pelo jogo.

6.2 REQUISITOS NÃO FUNCIONAIS

Contam como requisitos não funcionais deste projeto as seguintes características:

- **RNF-01:** O aplicativo deve ser executado em dispositivos móveis que utilizem o Sistema Operacional Android.
- **RNF-02:** Deve conter um conjunto de peças que simbolizam as 15 instruções aceitas pelo jogo: Função Estrela, Função Círculo, Função Triângulo, Virar à Esquerda, Virar à Direita, Mover, *Loop*, e números de dois a nove.
- **RNF-03:** O sistema deve utilizar alguma biblioteca ou framework de Visão Computacional para realizar a identificação das peças contidas em uma imagem como as instruções aceitas pelo jogo.
- **RNF-04:** Para a produção do aplicativo mobile, deve ser utilizada alguma Game Engine.
- **RNF-05** A arquitetura do sistema é constituída por cliente e servidor, com um aplicativo mobile, sendo o cliente, e um servidor sendo responsável pelo reconhecimento e processamento de de imagem.

6.3 TECNOLOGIAS UTILIZADAS

O início do processo de desenvolvimento deste projeto se deu através da escolha das ferramentas adequadas para auxiliar e facilitar o processo, além de garantir que as mesmas supririam todas as necessidades, com o objetivo de evitar atrasos ou dificuldades no desenvolvimento devido à uma escolha ineficiente. Nesta seção, descreve-se quais foram as ferramentas utilizadas, porque foram necessárias e como foi o processo de escolha destes recursos.

6.3.1 **GAME ENGINES**

Game engines ou, em português, motores de jogos, são *softwares* ou conjuntos de bibliotecas, que implementam diversas funcionalidades que simplificam o desenvolvimento de jogos. As bibliotecas associadas podem ser utilizadas para implementar comportamentos relativos à física do jogo, animações, áudio, interface gráfica, inteligência artificial e recursos de rede (BUENO, 2010).

O uso desse tipo de ferramenta permite que os desenvolvedores foquem mais nas regras e no enredo dos jogos do que na implementação de detalhes dos níveis mais baixos da aplicação. Algumas *engines* permitem inclusive que os desenvolvedores criem jogos sem implementar uma única linha de código (BUENO, 2010).

A primeira tarefa no desenvolvimento deste projeto foi encontrar um motor adequado que satisfizesse as necessidades deste trabalho e facilitasse o desenvolvimento do jogo. Atualmente existe uma grande quantidade de *Game Engines* que podem ser utilizadas para este fim.

Ao pesquisar sobre as ferramentas existentes, foi encontrada uma tabela com 161 *Game Engines* disponíveis no mercado (WIKIPEDIA). A tabela reúne informações sobre essas ferramentas, como a linguagem em que foram produzidas, quais linguagens se pode utilizar para desenvolver os jogos e as plataformas suportadas. Para filtrar as opções mais úteis, foram estabelecidos alguns critérios necessários para atender aos seguintes requisitos:

- A ferramenta deveria suportar o desenvolvimento de jogos voltados para a plataforma Android.
- Permitir a programação dos jogos em linguagens conhecidas pelos autores deste trabalho ou bem conhecidas pela comunidade em geral como, Java, JavaScript, Python, C# ou C++.
- Deveriam ser gratuitas ou *Open Source*.

Após esse filtro inicial restaram 25 *engines* que passaram por um novo processo de verificação, levando em conta os seguintes critérios:

- **Engajamento da comunidade:** como a ferramenta é utilizada pelos usuários, o engajamento da comunidade nos fóruns sobre o produto e a quantidade de tutoriais e material introdutório produzido pelos usuários ou pelos próprios desenvolvedores da ferramenta tornou-se um fator relevante.
- **Documentação:** existência e detalhamento da documentação sobre a ferramenta.
- **Facilidade de uso:** recursos que as ferramentas disponibilizam para auxiliar no desenvolvimento dos jogos e estimativa da dificuldade relativa à produção dos mesmos.

Após esta segunda etapa de filtro, 8 *engines* foram selecionadas para o levantamento dos pontos fortes e fracos de cada uma descritos no quadro 1:

Quadro 1 - Análise de Game Engines (continua)

Engine	**Vantagens**	**Desvantages**
Xenko	- Documentação completa, bem estruturada e de fácil entendimento. - Gratuito para estudantes e pequenas empresas. - Possui um editor que facilita a criação dos jogos	- Comunidade pouco engajada - Pouco material encontrado relativo à manipulação de recursos de dispositivos móveis, como a câmera - Depende de outros *softwares* para que o jogo seja portado para a plataforma Android
Unity	- Ferramenta traduzida para o português. - Comunidade grande e engajada. - Possui um plano gratuito para estudantes e entusiastas. - Documentação completa, bem estruturada e de fácil entendimento - Possui um editor que facilita a criação dos jogos	- Não fornece acesso direto ao código do aplicativo gerado
Unreal	- Ferramenta traduzida para o português. - Comunidade grande e engajada. - Possui um plano gratuito para estudantes e entusiastas. - Documentação completa, bem estruturada e de fácil entendimento - Possui um editor que facilita a criação dos jogos	- Não fornece acesso direto ao código do aplicativo gerado.
V-play	- Possui alguns *templates* de jogos prontos. - Possui um editor que facilita a criação dos jogos. - Possui uma versão gratuita	- Não possuir meios nativos para acessar os recursos dos dispositivos *mobile*. - Comunidade pequena. - Pouco material de aprendizado.
Cocos2d	- Ferramenta *Open Source* Comunidade bastante ativa e engajada - Grande quantidade de tutoriais disponíveis - Possui um editor que facilita a criação dos jogos	- Documentação dispersa

Quadro 1 - Análise de Game Engines (conclusão)

Engine	**Vantagens**	**Desvantages**
Phaser	- Documentação completa, bem estruturada e de fácil entendimento; - Possui diversos canais de comunicação com a comunidade, como através das ferramentas Slack e Discord. - Possui um editor que facilita a criação dos jogos	- Esta é uma *engine* para jogos em HTML 5, que embora possam ser executadas em dispositivos *mobile*, não geram aplicativos instaláveis. - Não fornece acesso a alguns recursos dos dispositivos *mobile*, como a Câmera.
PlayCanvas	- Ferramenta *Open Source* - Possui um editor que facilita a criação dos jogos - Possui diversos tutoriais de aprendizado na página oficial da engine	- Documentação pequena e pouco detalhada. - Esta é uma engine para jogos em HTML 5, que embora possam ser executadas em dispositivos *mobile*, não geram aplicativos instaláveis. - Não fornece acesso a alguns recursos dos dispositivos *mobile*, como a Câmera.
PlayN	- *Engine* desenvolvida pela Google - *Open Source*	- Não possui Editor - Documentação ruim, mal organizada e insuficiente - Pouco utilizada, com uma comunidade bastante reduzida. - Esta é uma *engine* para jogos em HTML 5, que embora possam ser executadas em dispositivos *mobile*, não geram um arquivo instalável.

Fonte: Os autores (2018)

Destas ferramentas, aquelas que geram jogos em HTML5 foram excluídas, pois, apesar de ser possível acessar estes jogos através de dispositivos móveis, as ferramentas não geram aplicativos instaláveis, o que poderia prejudicar o acesso a alguns recursos dos dispositivos, como a câmera, fundamental para o funcionamento desta aplicação.

Outro fator excludente foi a falta ou má estruturação das documentações e a baixa adesão às ferramentas. A análise sobre esses requisitos foi feita empiricamente durante o processo de busca por informações sobre estas e*ngines*. Esses fatores foram considerados como excludentes porque poderiam dificultar o processo de aprendizado sobre a ferramenta ou a resolução de problemas que fossem encontrados durante o desenvolvimento.

Com essa nova análise, foram selecionadas as duas melhores *engines* que atendiam os critérios/requisitos descritos: Unity e Unreal Engine. Essas duas estão frequentemente presentes nos rankings de melhores ferramentas para o

desenvolvimento de jogos(ELHADY, 2017; FLORIAN, 2018).

O Unity, atualmente, domina o mercado de games *mobile*, com o domínio de cerca de mais de 50% dos jogos voltados para esta plataforma (UNITY TECHNOLOGIES) e vem sendo bastante utilizado por produtoras de jogos independentes. É uma ferramenta mais indicada para desenvolvedores iniciantes, visto que contém uma grande quantidade de material para aprendizado e recursos como *plugins* e *sprites* bastante acessíveis (CHRISTOPOULOU; XINOGALOS, 2017).

A Unreal é mais utilizada para jogos profissionais, fornecendo maior controle sobre o código fonte da *engine* e tendo como foco o mercado de jogos AAA (jogos que possuem maior qualidade, desempenho e possuem um caráter mais comercial). É mais indicada para desenvolvedores experientes e exige um *hardware* mais robusto para seu funcionamento (CHRISTOPOULOU; XINOGALOS, 2017).

Embora o Unreal possua melhor desempenho e seja voltado para jogos profissionais, o Unity se adequa à realidade de um jogo simples e voltado para a plataforma móvel, como o desenvolvido neste projeto. Outro fator determinante para a escolha foi o engajamento da comunidade de desenvolvimento. O Unity possui uma vasta quantidade de fóruns, tutoriais e vídeos ensinando o seu funcionamento e demonstrando o quão fácil é criar um jogo com a plataforma em pouco tempo, algo crucial para este projeto, tendo em vista a baixa experiência dos autores em desenvolvimento de jogos e o tempo limitado de produção.

6.3.1.1 UNITY

O Unity é uma ferramenta robusta, completa e simples, capaz de produzir, segundo informado no próprio site da ferramenta (UNITY TECHNOLOGIES) , jogos para mais de 25 plataformas diferentes. Para este projeto foi utilizado a versão 2017.3.1.1f1 fazendo uso do plano gratuito da plataforma, que possui além deste, dois outros planos pagos.

O recurso mais utilizado neste projeto foi o Editor Unity, um *software* disponível para Windows, Mac e Linux, que fornece uma série de ferramentas úteis para o desenvolvimento de jogos, como editores de interfaces com o usuário, ferramentas de *script* para a linguagem C# e gerenciadores de animações.

Outra plataforma bastante útil fornecida pela Unity é a Unity Asset Store, onde é possível encontrar uma vasta coleção de *sprites*, *scripts*, e *plugins* produzidos pela comunidade e distribuídos de forma gratuita ou paga.

6.3.2 **VISÃO COMPUTACIONAL**

A cada fase, o jogador resolve o problema proposto organizando suas peças e capturando uma fotografia das instruções. Para identificar os comandos indicados na fotografia, foram utilizadas técnicas da visão computacional.

Segundo MILANO e HONORATO (apud WIZBICKI e BATTISTI, 2009, p.1) “A Visão Computacional é a área da computação que se dedica a desenvolver teorias e métodos voltados à extração automática de informações úteis contidas em imagens”, ou seja, caracteriza-se pelo desenvolvimento de sistemas que possam, ao receber uma imagem, analisá-la e extrair dados e informações relevantes para o contexto da aplicação.

As técnicas de Visão Computacional geralmente começam com aplicações de algoritmos de processamento de imagem como, por exemplo, a redução de ruídos, redimensionamento e melhorias de contraste (MARENGONI; STRINGHINI, 2009). Outros procedimentos podem ser aplicados à imagem pré-processada com o objetivo de extrair informações relevantes, como as operações de segmentação, que consistem em particionar a imagem em regiões ou objetos distintos , bem como a classificação, que é o reconhecimento dos elementos identificados na imagem (MARENGONI; STRINGHINI, 2009).

Os procedimentos necessários para o processo de Visão Computacional costumam ser complexos (WIZBICKI; BATTISTI), e para que esta etapa de desenvolvimento fosse feita da forma adequada, foi necessário identificar ferramentas que facilitem a execução destes processos.

Existe uma grande quantidade de ferramentas de Visão Computacional disponíveis no mercado, variando de soluções de uso mais simples que executam todos os processos necessários de forma automática e sem esforço do desenvolvedor, como o Google VIsion API (GOOGLE CLOUD) e IBM Watson (IBM), até ferramentas mais cruas, como bibliotecas de desenvolvimento que, embora contenham uma série de algoritmos já implementados, deixam a cargo do desenvolvedor a tarefa de implementar o sistema de reconhecimento como o OpenCV (OPENCV TEAM) e Tensor Flow (TENSOR FLOW).

O quadro 2 apresenta as vantagens e desvantagens destas ferramentas, pré-selecionadas:

Quadro 2 - Ferramentas de Visão Computacional

Nome	Vantagens	Desvantagens
Google Vision API	- *Software* produzido pela Google; - Documentação completa, bem estruturada e de fácil entendimento; - Integração nativa com a plataforma Android; - Capacidade de reconhecimento de textos, códigos de barra, e classificação de imagens; - Possui um plano gratuito de acesso, até o limite de 1000 requisições ao mês; - É de fácil utilização, basta fazer uma requisição à API com a imagem a ser processada.	- Poucas fontes de aprendizado. Não é uma ferramenta muito utilizada para o reconhecimento de objetos e por isso faltam tutoriais; - Reconhece muitos elementos das imagens, mas não permite customização, retornando para o usuário tags pré-determinadas de itens reconhecidos; - Não oferece controle sobre como a imagem é processada.
OpenCv	- Ferramenta *Open Source*; - Comunidade bastante ativa e engajada. Os fóruns de dúvidas e discussões são bastante movimentados e há uma grande quantidade de tutoriais criados pelos usuários; - Documentação completa, bem estruturada e de fácil entendimento; - Conta com diversos algoritmos já implementados para a transformação das imagens e para a implementação de aprendizado de máquina.	- Exige uma grande curva de aprendizado para utilizar a biblioteca.
Tensor Flow	- Biblioteca criada pela Google; - *Open source*; - Voltada para o desenvolvimento de *machine learning*; - Possui uma grande quantidade de modelos de aprendizado já prontos.	- Não é estritamente voltada para Visão Computacional, embora também possa ser utilizada para isso; - Exige uma grande curva de aprendizado para utilizar a biblioteca.
IBM Watson	- Ferramenta produzida pela IBM e uma das ferramentas de Inteligência Artificial mais utilizadas atualmente; - Fácil utilização, bastando enviar uma requisição para a API; - Permite o treinamento da API para reconhecer elementos específicos nas imagens com *tags* customizadas.	- Não oferece controle sobre como a imagem é processada.

Fonte: Os autores (2018)

Diante das opções disponíveis, optou-se por fazer alguns testes utilizando a ferramenta Google Cloud Vision. A princípio, essa aparentou ser uma excelente alternativa, pois se tratava de uma ferramenta criada por uma das maiores empresas

de tecnologia do mundo, que possui integração com a plataforma Android e que é de fácil e rápida utilização, dependendo apenas de requisições para uma API e, portanto, não exigindo uma curva de aprendizado tão grande.

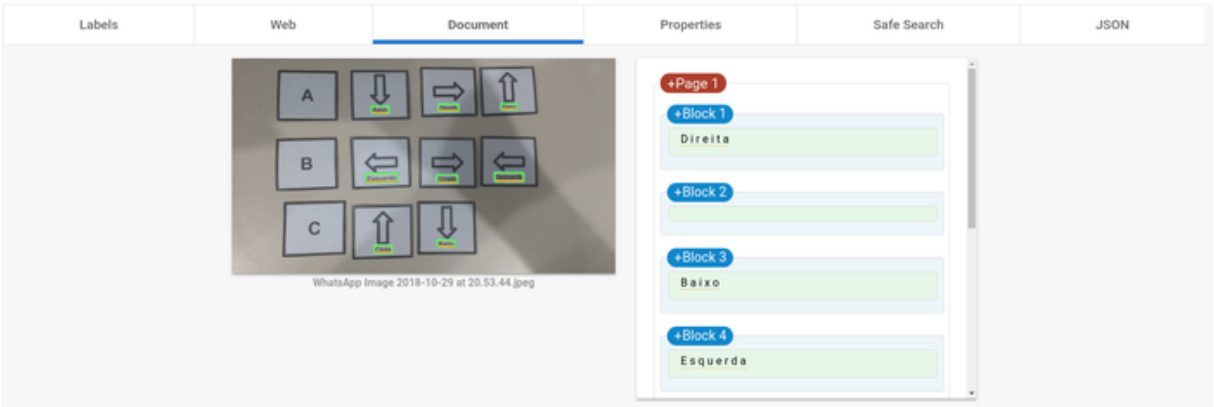
O site da ferramenta disponibiliza um espaço em que é possível testar a API, permitindo anexar uma foto para passar pelo processo de Visão Computacional e exibindo a resposta recebida. A API permite selecionar o algoritmo procura na imagem, como o reconhecimento de elementos textuais, rostos, conteúdo impróprio e marcadores.

Esse recurso foi utilizado para verificar qual algoritmo atenderia às necessidades do projeto. Como principal ponto, a ferramenta deveria reconhecer, de alguma forma, cada uma das peças dispostas na imagem e a ordem da disposição das peças.

Dos modos de requisição oferecidos pela ferramenta, apenas o reconhecimento de texto fornece as coordenadas dos elementos identificados na imagem, o que permitiria fazer um processamento sobre o resultado para ordenar as peças. Sendo assim, foram inseridos alguns textos junto aos ícones nas peças do jogo e capturadas algumas fotos de algoritmos montados com elas. Essas imagens foram carregadas no site e as respostas analisadas.

Os resultados obtidos não foram considerados satisfatórios, pois a ferramenta não foi capaz de reconhecer corretamente os textos colocados nas peças, reconhecendo palavras incorretas ou até mesmo deixando de identificar o texto em algumas das peças. A figura 20 exemplifica um destes casos em que a ferramenta de testes não consegui reconhecer corretamente as peças.

Figura 20 - Teste do Google Vision API



Fonte: Os autores (2018)

Por ser uma aplicação fechada, o Google Vision API não permite customização na forma como as imagens e seus elementos são processados e identificados, impedindo que melhoramentos ou direcionamentos para casos

específicos fossem feitos. A junção destes fatores fez com que essa ferramenta e aquelas que funcionam de forma similar, fossem descartadas.

Foi necessário partir para uma outra abordagem, buscando utilizar uma ferramenta que disponibilizasse um maior controle sobre o processo de Visão Computacional e que permitisse realizar alterações em caso de resultados incompatíveis com o esperado.

Embora algumas das outras opções também forneçam algum tipo de customização na classificação dos objetos reconhecidos nas imagens, como o IBM Watson, nenhuma delas fornece opções para o processamento das fotografias, como a alteração de cores, tamanhos e reconhecimento de contornos. Dentre as opções identificadas, a que mais se adequava a essas necessidades era o OpenCV.

6.3.2.1 OPENCV

O OpenCV é uma biblioteca *open source* de Visão Computacional e aprendizado de máquina. Foi idealizada para fornecer uma infraestrutura comum para aplicações de Visão Computacional e para ser uma alternativa que permita alterações de código por parte dos negócios que a utilizarem (OPENCV TEAM).

A biblioteca conta atualmente com mais de 2500 algoritmos de Visão Computacional e *Machine Learning*, com mais de 14 milhões de *downloads* (OPENCV TEAM) , formando uma comunidade expressiva e engajada.

O OpenCV pode ser utilizado nas linguagens C++, Python, Java e MATLAB. Para este trabalho, foi utilizada a versão 3.3.1 da biblioteca na linguagem Python. Para utilizar tal ferramenta, é necessário instalar a biblioteca que já possui implementações de algumas técnicas de processo de Visão Computacional, vista na seção 6.3.2, e fica a cargo do desenvolvedor realizar todas as chamadas e tratamentos da imagem, utilizando os algoritmos que julgar adequado para conseguir o resultado desejado.

6.4 DESENVOLVIMENTO

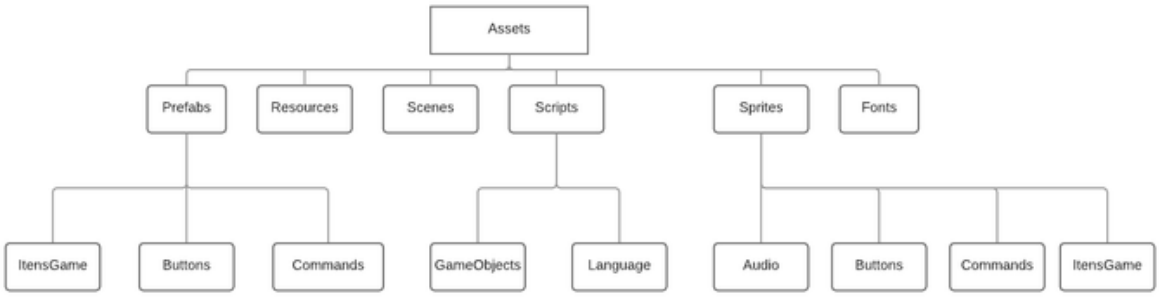
Após identificar e escolher as ferramentas utilizadas no desenvolvimento deste trabalho, foi dado início a implementação do projeto. Neste capítulo descreve-se como se deu o desenvolvimento do aplicativo, do módulo de Visão Computacional e a integração de ambos.

6.4.1 **DESENVOLVIMENTO DO APLICATIVO**

O aplicativo *mobile* que representa o jogo proposto foi feito utilizando a ferramenta Unity, com o auxílio do Unity Editor. Os scripts foram escritos em C#, linguagem padrão do Unity.

A Figura 21 representa a estrutura de diretórios da qual o aplicativo é constituído:

Figura 21 - Diagrama de classes do aplicativo



Fonte: Os autores (2018)

A pasta “*Prefabs*” contém o que o Unity chama de *Prefabs*. Esses elementos são, basicamente, *Game Objects* que podem ser criados pelo usuário para serem utilizados em diferentes cenas do jogo, mas mantendo as mesmas propriedades e comportamentos, como se fosse um modelo (UNITY TECHNOLOGIES).

Um *Game Object*, por sua vez, é um objeto fundamental do Unity, que pode representar personagens, componentes de cenário e propriedades. Este elemento serve como um container, no qual podem ser adicionadas diversas propriedades como movimentação, luz, gravidade, etc (UNITY TECHNOLOGIES).

Os *Prefabs* estão divididos na nossa estrutura entre “*ItensGame*” que abriga os *Game Objects* que compõem o cenário do desafio, como o chão, obstáculos, a nave e os itens coletáveis; “*Buttons*” que abriga os botões como de volume, ajuda, entre outros; “*Command*” que abriga os *Game Objects* que simbolizam as instruções a serem realizadas.

Na pasta “Resources” ficam os arquivos em formato JSON que contém as fases do jogo conforme descrito na seção 6.4.1.1.

Na pasta “*Scenes*” estão contidas as cenas do jogo. No Unity, uma cena representa uma tela do jogo (UNITY TECHNOLOGIES). Neste trabalho foram utilizadas três cenas diferentes, sendo a primeira a tela inicial do jogo, a segunda é a tela de escolha de fases e a terceira sendo o nível em sí, cujos elementos são carregados dinamicamente baseado no arquivo JSON da fase escolhida.

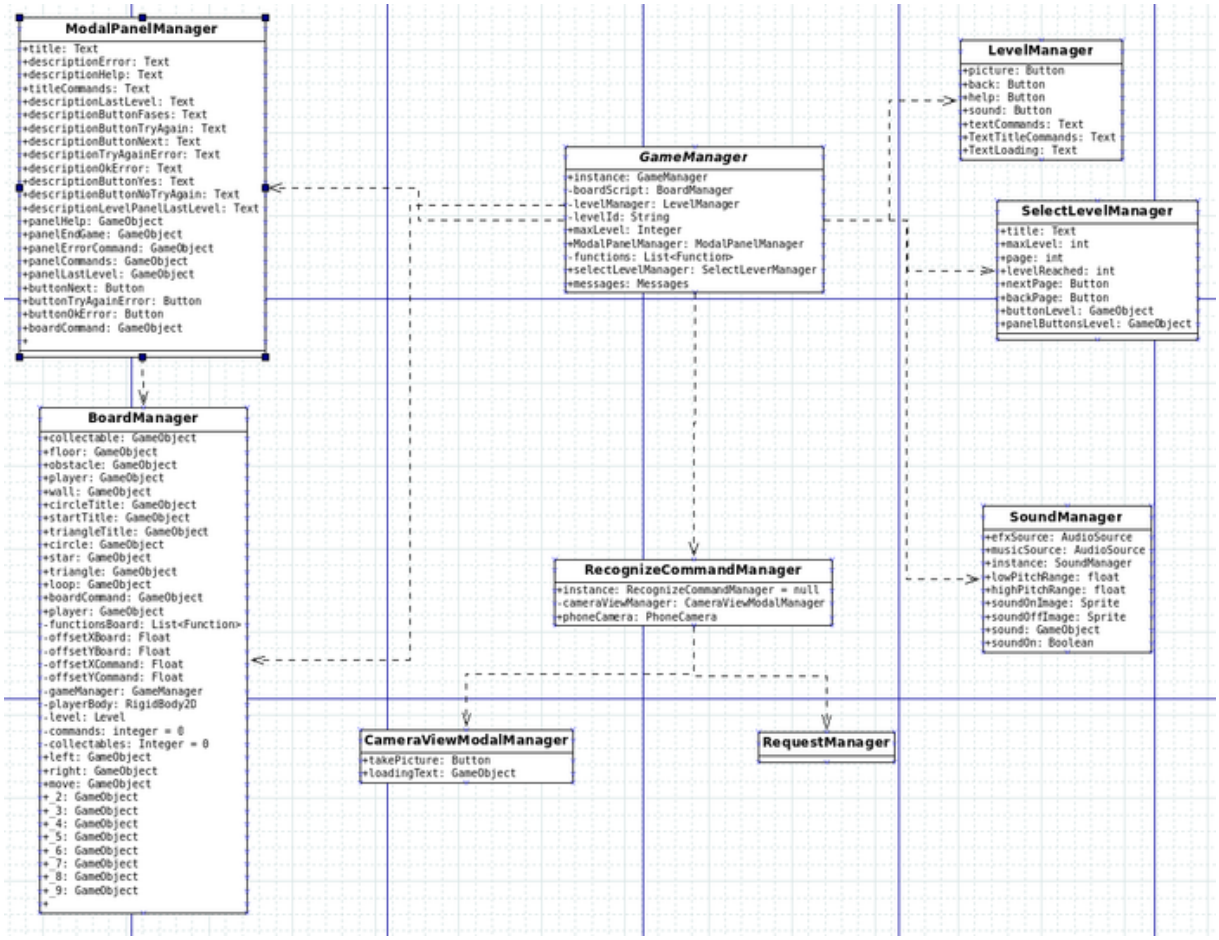
Na pasta “*Sprites*” estão contidos os elementos gráficos do jogo. *Sprites* são, conceitualmente, objetos gráficos 2D (UNITY TECHNOLOGIES) designados a serem parte de uma cena maior (TECHTERMS, 2012). As *Sprites* podem ser utilizadas, por exemplo, para representar visualmente um personagem do jogo ou qualquer outro elemento do cenário.

Neste projeto tem-se como exemplo os ícones que representam os botões do jogo, o foguete que se move pelo tabuleiro de acordo com as instruções definidas e até mesmo o chão pelo qual ele se move. As *sprites* e o design da interface do aplicativo foram feitos por Kanarek Brunel (2018) de forma gratuita. Nesta pasta também estão armazenados alguns arquivos de áudio que são utilizados no jogo e disponibilizadas no tutorial 2D UFO Tutorial(UNITY TECHNOLOGIES).

Dentro da pasta “*Scripts*” encontra-se a pasta “*Languages*”. Dentro desta pasta está a classe “*Messages*” que possui métodos abstratos representando todos os textos contidos no jogo. Para inserir elementos textuais no jogo, todas as classes consultam o “*GameManager*” que controla qual o idioma vigente nas configurações do usuário, e cria uma instância desse idioma através de uma classe que implementa a classe “*Messages*” e contém os textos na linguagem definida. Assim para que o jogo aceite um novo idioma é apenas necessário criar um novo *script* que implemente os métodos abstratos e instanciá-lo no “*GameManager*”

Na pasta “*Scripts*” também estão armazenados outros *scripts* de código que são utilizados pela aplicação e que executam as principais funções do jogo. No diagrama 1 é possível ver um diagrama de classes com os principais Scripts.

Diagrama 1 - Diagrama com as principais classes do aplicativo



Fonte: Os autores (2018)

Dentro da pasta “*Scripts/GameObjects*” estão os *scripts* responsáveis por controlar ações sobre os *GameObjects* da cena, como, por exemplo, “*SoundManager*”, que controla a funcionalidade de ligar e desligar o som conforme ação do usuário, “*Collectable*”, que tem função de realizar a animação dos itens coletáveis da fase, entre outros.

Os demais *scripts* possuem uma maior lógica e atribuição de papéis, como o “*GameManager*” responsável por controlar o fluxo de informações, o “*BoardManager*” responsável pela criação e movimentação do tabuleiro e *board* de comandos, o “*RecognizeCommandManager*”, que através da foto tirada pela câmera gerencia as chamadas para descobrir quais são os comandos contidos na mesma para a execução da fase e o “*RequestManager”* responsável pela comunicação com o servidor da aplicação de reconhecimento de imagem descrito na seção 6.4.3.

6.4.1.1 CARREGAMENTO DOS NÍVEIS E MONTAGEM DO TABULEIRO

Com o objetivo de facilitar a criação de novos desafios, foi definido um esquema pelo qual as fases podem ser carregadas de forma dinâmica a partir de um arquivo JSON, seguindo as especificações definidas para o arquivo, sendo possível criar uma nova fase em poucos minutos. Na figura 22 é possível ver a especificação do arquivo JSON.

Figura 22 - Modelo de arquivo JSON


```
{
        "name": "Level 1",
        "levelID": "1",
        "author": "Everton/Yuri",
        "difficulty": 1,
        "maxCommands": 40,
        "maxCommandsUse": 40,
        "playerDirection": "up",
        "board": {
                "data": [
                        {
                                "positionX": 0,
                                "positionY": -1,
                                "objectType": "Player"
                        },
                        {
                                "positionX": 0,
                                "positionY": 0,
                                "objectType": "Floor"
                        },
                        {
                                "positionX": 0,
                                "positionY": 1,
                                "objectType": "Collectable"
                        }
                ]
        }
}
```


Fonte: Os autores (2018)

O arquivo contém atributos para identificação da fase, como o nome, seu ID, o autor, a dificuldade, a quantidade máxima de comandos que o usuário pode utilizar para montar a solução e a quantidade máxima de movimentos que podem ser realizados na fase. O atributo *playerDirection* indica a direção inicial para a qual o foguete aponta. O atributo *board* recebe uma lista de objetos, cada um com uma posição X e Y, que identifica a posição do mesmo no tabuleiro, e o tipo do objeto que ele pode ser: Chão, Coletável, Jogador, Parede ou Obstáculo.

Após a criação do arquivo, pode-se criar um botão na segunda cena que, quando clicado, informa ao “*GameManager*” o ID do level, sendo este o nome do arquivo JSON. O “*GameManager*” repassa a informação ao “*BoardManager*” que consegue ler o arquivo e montar o tabuleiro na interface do usuário, através do

método *boardSetup()*, conforme os dados informado no atributo de *board* descrito acima, que contém as informações dos objetos.

6.4.1.2 FOTOGRAFAR E EXECUTAR INSTRUÇÕES

Ao clicar no botão de “Câmera”, a classe “*RecognizeCommandManager*” é chamada através do método *takePictureClick()*, que utiliza o script “*PhoneCamera*” para acessar o dispositivo de câmera do celular. Ao tocar novamente no botão de “Câmera” o método *TakePhoto()* é chamado e grava os bytes dos pixels da câmera. Através desses bytes é realizada, de forma assíncrona, uma requisição ao servidor que abriga o módulo de visão computacional através da classe “*RequestManager*” pelo método *Request()*.

Código 1 - métodos de captura de imagem e requisição

```
public void takePictureClick()
{
    cameraViewManager.loading();
    byte[] bytes = phoneCamera.TakePhoto();
    request(bytes);
}

public void request(byte[] bytes)
{
    StartCoroutine(RequestManager.Request(bytes, this));
}
```

Fonte: Os autores (2018)

Então o “*RequestManager*” monta a requisição, informando o DNS da máquina onde está a aplicação de reconhecimento da imagem descritas nas seções 6.4.2 e 6.4.3, e transforma os bytes capturados em um conjunto de caracteres através do método Base64, utilizando o String resultante como conteúdo da requisição. Quando o servidor retorna à resposta da requisição, é chamada novamente a classe “*RecognizeCommandManager*” através do método *response()*.

O método *response()* por sua vez é responsável por desativar o dispositivo de câmera do usuário e, caso a resposta do servidor seja um erro, apresentar ao usuário um modal com a opção de tirar uma nova foto.

Se a resposta do servidor ocorreu de maneira correta, ela é encaminhada para o método *convertToCommand()* da classe “*RecognizeCommandManager*”. Esse método tem como função converter a resposta do servidor para objetos que representam cada uma das instruções e funções a serem compreendidas pelo aplicativo. Além disso, também é feita uma verificação sobre a sintaxe dos

comandos conforme descrito na seção 5.1, indicando o erro, do contrário o método *recognizeCommand()* da classe “*GameManager*” é chamado.

Após o reconhecimento dos comandos é apresentado um modal com comandos identificados e pedindo uma confirmação de que o reconhecimento teve sucesso, como mostrado na figura 23.

Figura 23 - Tela de confirmação dos comandos identificados

Fonte: Os autores (2018)

No caso de os comandos terem sido identificados de forma incorreta, o usuário pode clicar em “Não, vou tentar de novo” e ele será encaminhado para capturar outra fotografia, repetindo todo o processo já descrito.

Caso o usuário informe que as funções reconhecidas estão corretas, o método *doCommands()* do script "*BoardManager"* é chamado e apresenta, no painel da parte inferior da tela, os comandos identificados e passa por parâmetro a primeira linha de comando para ser executada a cada frame pelo método *DoFunction()*.

Código 2 - Trecho do método DoFunction()

```
private IEnumerator DoFunction(Function function)
{
    previous = null;
    foreach (Command command in function.Commands)
    {
        animationCommand(command);
        if (!endGame)
        {
            endGame = gameManager.checkEndGameCommand();
            switch (command.EnumCommand)
            {
                case EnumCommand.MOVE:
                    yield return StartCoroutine(Move());
                    break;
                case EnumCommand.CIRCLE:
                    yield return new WaitForSeconds(1f);
                    animationCommand(null);
                    yield return StartCoroutine(DoFunction(functionsBoard[indexCircle]));
                    break;
                case EnumCommand.STAR:
                    yield return new WaitForSeconds(1f);
                    animationCommand(null);
                    print(indexStar);
                    yield return StartCoroutine(DoFunction(functionsBoard[indexStar]));
                    break;
                case EnumCommand.TRIANGLE:
                    yield return new WaitForSeconds(1f);
                    animationCommand(null);
                    yield return StartCoroutine(DoFunction(functionsBoard[indexTriangle]));
                    break;
                case EnumCommand.LOOP:
                    yield return StartCoroutine(Loop(command));
                    break;
                case EnumCommand.LEFT:
                    yield return StartCoroutine(Turn(PlayerDirection.LEFT));
                    break;
                case EnumCommand.RIGHT:
                    yield return StartCoroutine(Turn(PlayerDirection.RIGHT));
                    break;
                case EnumCommand.CIRCLE_TITLE:
                    commands--;
                    break;
                case EnumCommand.STAR_TITLE:
                    commands--;
                    break;
                case EnumCommand.TRIANGLE_TITLE:
                    commands--;
                    break;
            }
```

Fonte: Os autores (2018)

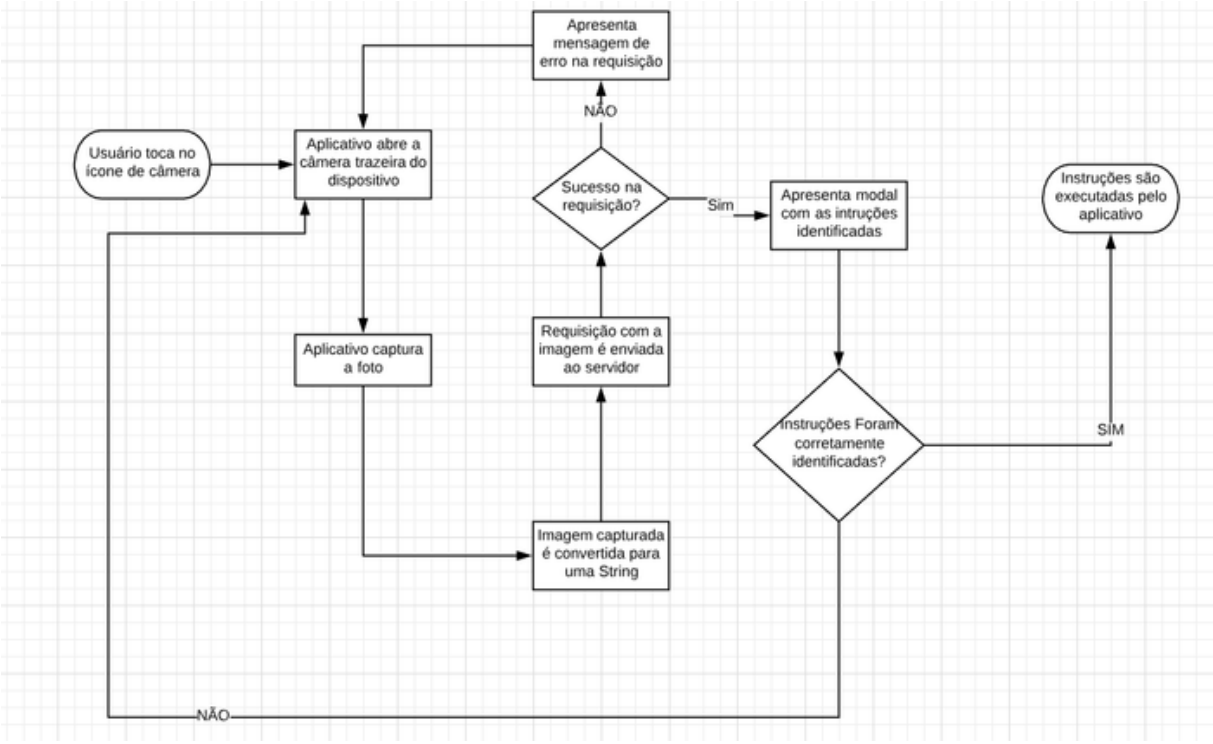
O método *DoFunction()* da classe "*BoardManager*" tem como objetivo

executar uma única função, ou seja, uma linha de comandos fotografada pelo usuário. Para cada comando contido dentro da função, é executado o método *animationCommand()*, que realiza a animação do comando a ser executado, apresentando para o usuário visualmente qual comando está sendo executado no momento.

O *DoFunction()* também verifica se, com o comando executado, o jogo atingiu o limite de comandos para a fase correspondente. Assim, a execução dos comandos é interrompida e a derrota é comunicada ao usuário. Por fim, esse método verifica a ação do comando que deve executar: Mover o foguete, virar em uma direção, iniciar um laço de repetição para algumas instruções ou chamada de função. Se o comando é uma função, o método *DoFunction()* é chamado recursivamente passando a função destino.

O Fluxograma 1 apresenta de forma simplificada o processo de reconhecimento e execução das instruções descrito.

Fluxograma 1 - Fluxo de captura da imagem, identificação de instruções e execução.



Fonte: Os autores (2018)

6.4.2 **DESENVOLVIMENTO DO MÓDULO DE VISÃO COMPUTACIONAL.**

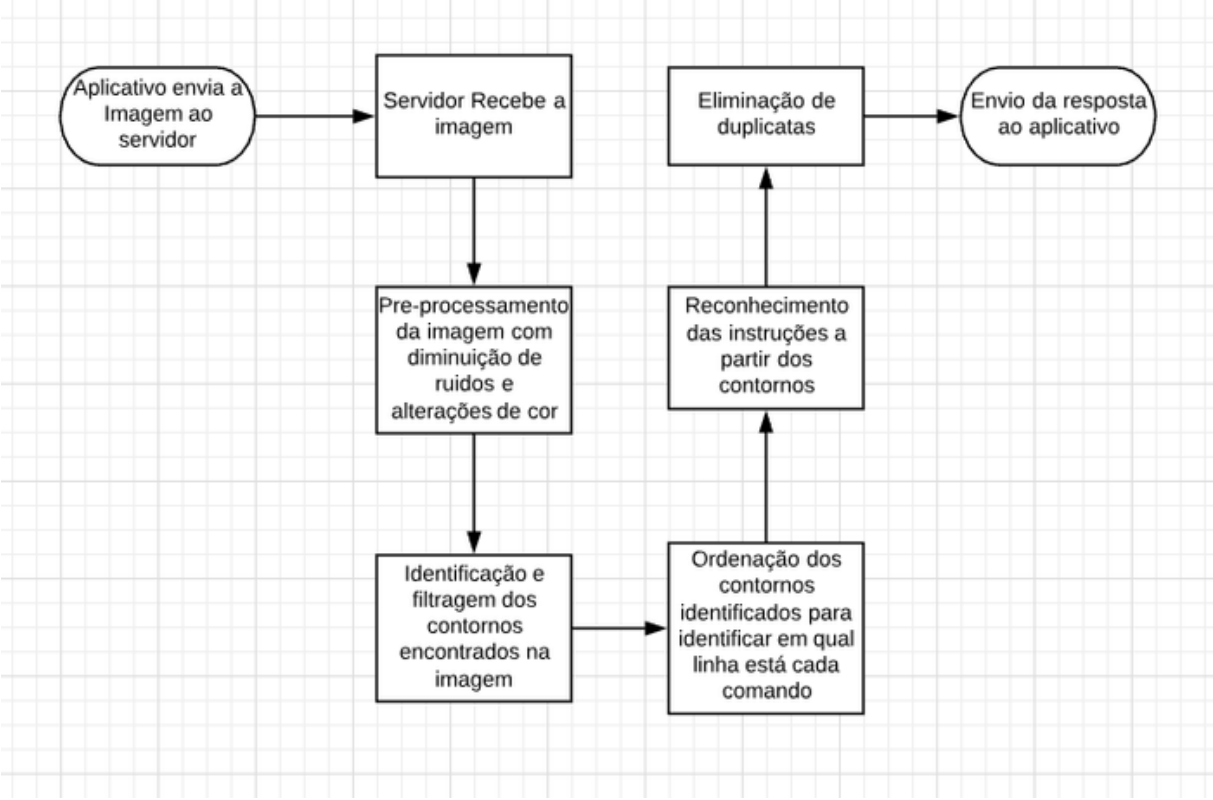
O módulo de reconhecimento de imagem foi desenvolvido com base em uma aplicação feita para o reconhecimento de cartas de baralho (EDJEELETRONICS,

2017). Essa aplicação é capaz de, em tempo real, ao filmar cartas dispostas sobre um fundo preto, reconhecer o naipe e o número da carta.

O projeto possui a seguinte estrutura: o arquivo "*Command.py"*, que contém o *script* responsável por realizar o pré-processamento da imagem e reconhecer os comandos; o arquivo "*CommandTestDetector.py"* tem como objetivo rodar todo o conjunto de teste; a pasta "*Commands_imgs"* que contém as imagens base das instruções que serão reconhecidas pelo sistema em desenvolvimento; a pasta "*test"* que possui imagens utilizadas para testar e validar o algoritmo de Visão Computacional.

O fluxograma 2 apresenta uma forma simplificada do processo de reconhecimento das peças contidas na imagem enviada pelo usuário através do aplicativo do jogo.

Fluxograma 2 - Fluxo de identificação das instruções



Fonte: Os autores (2018)

Para processar e reconhecer os comandos contidos em uma fotografia são utilizadas funções escritas no arquivo "*Command.py*". O início se dá pela chamada da função *preprocess_image()* que recebe uma imagem e aplica alguns filtros para corrigir grandes variações de luz e cor.

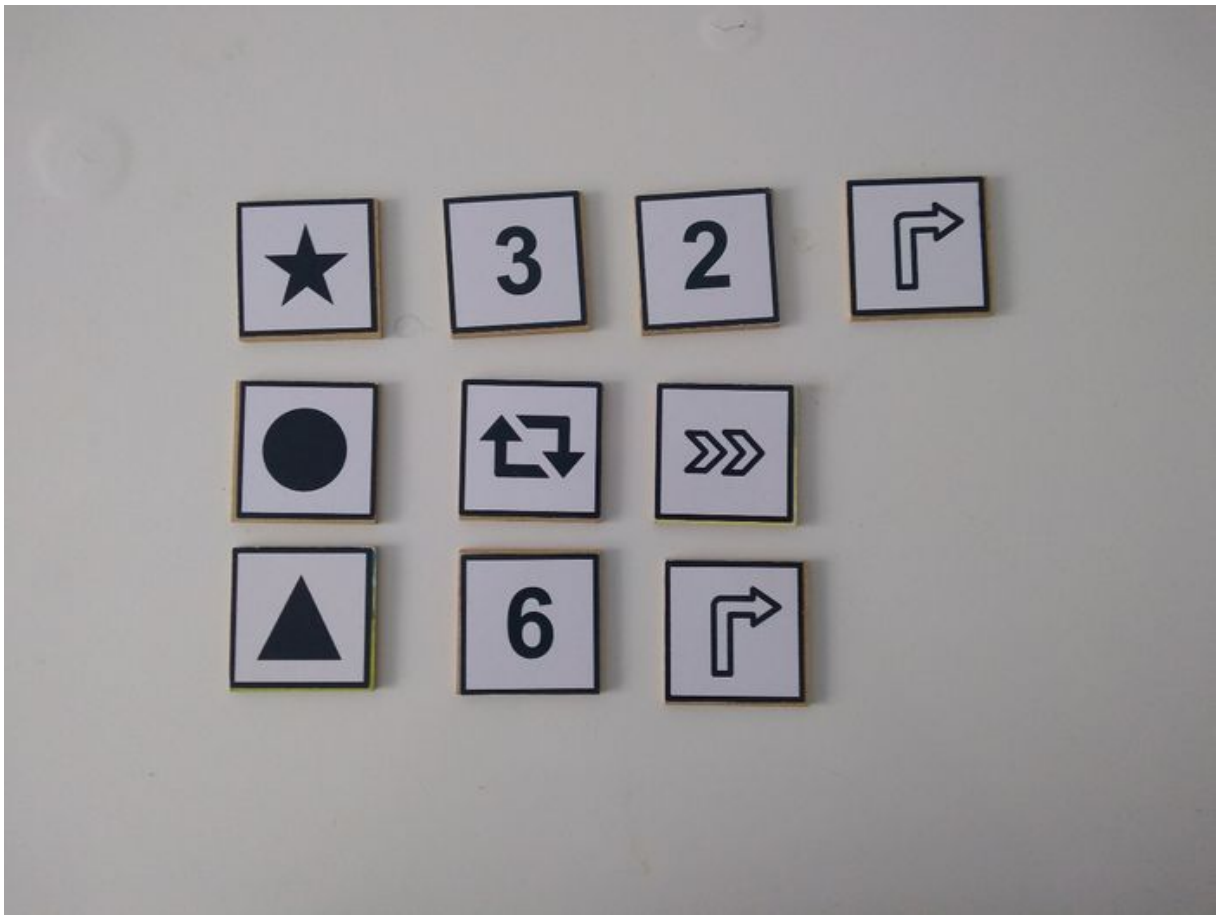
Código 3 - método preprocess_image()

```
53  def preprocess_image(image):
54      gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
55      blur = cv2.GaussianBlur(gray,(5,5),0)
56      thresh = cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,11,2)
57      return thresh
```

Fonte: Os autores (2018)

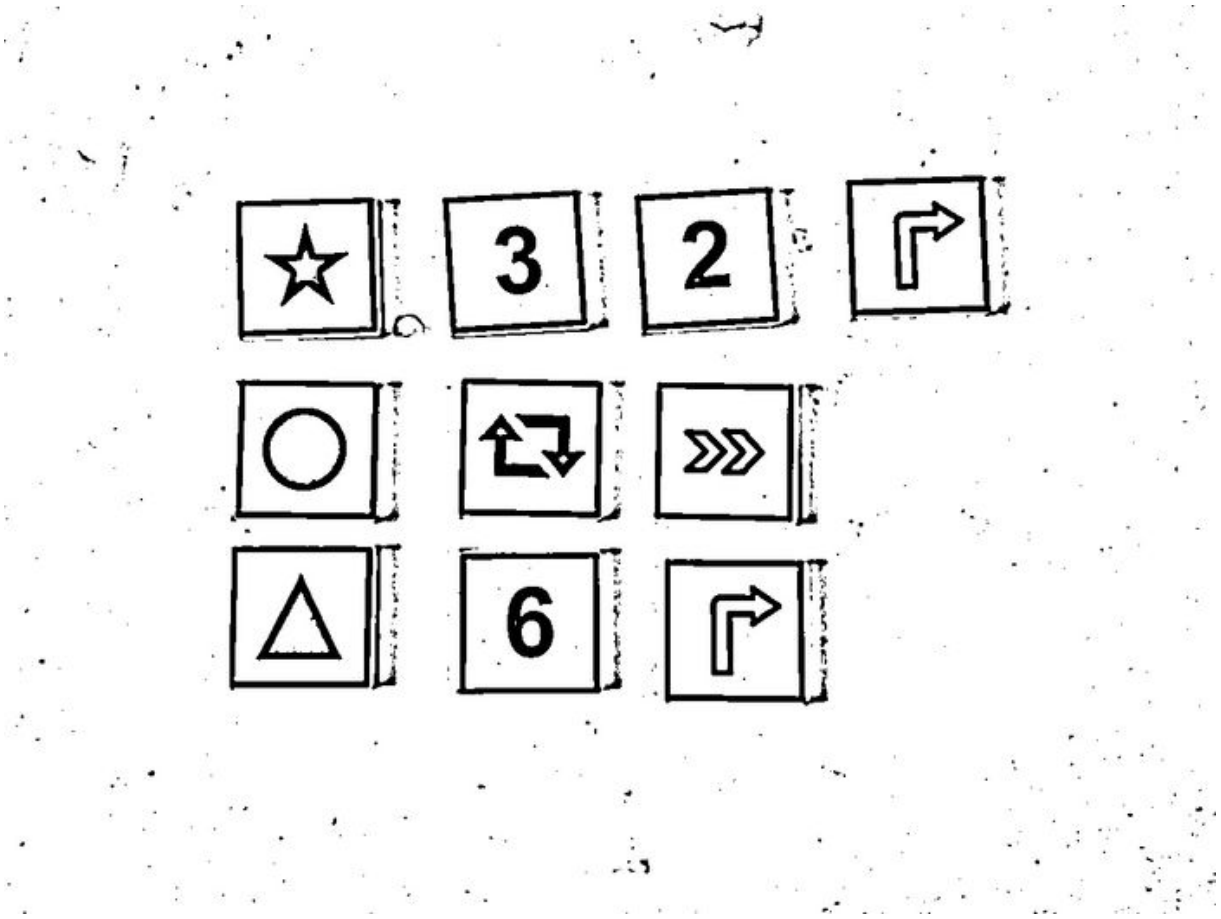
A imagem recebida tem suas cores transformadas em uma escala de cinza e é novamente processada para remover ruídos e pixels de alta frequência . A imagem resultante passa pela função *cv2.adaptativeThreshold()* que transforma os pixels da imagem em preto ou branco, com base na média dos pixels a sua volta. As figuras 24 e 25 mostram a fotografia antes e depois de passar por este processo.

Figura 24 - Imagem não processada



Fonte: Os autores (2018)

Figura 25 - Resultado do processamento da imagem



Fonte: Os autores (2018)

A imagem resultante do pré-processamento é submetida a função *find_cnts_commands()*.

Código 4 - Método find_cnts_commands()

```
59  def find_cnts_commands(thresh_image):
60      dummy,cnts,hier = cv2.findContours(thresh_image,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
61      cnts_return = []
62      for i in range(len(cnts)):
63          size = cv2.contourArea(cnts[i])
64          peri = cv2.arcLength(cnts[i],True)
65          approx = cv2.approxPolyDP(cnts[i],0.01*peri,True)
66          if len(approx) == 4 and (size < COMMAND_MAX_AREA) and (size > COMMAND_MIN_AREA):
67                  cnts_return.append(cnts[i])
68      return cnts_return, len(cnts), len(cnts_return)
```

Fonte: Os autores (2018)

Neste método é utilizada a função *cv2.findContours()* disponibilizada pela biblioteca OpenCV e que é a responsável por identificar os contornos na imagem. O

que esta função faz é, basicamente, ligar os pixels próximos que possuem cores e intensidades parecidas. Como a imagem fornecida está em preto e branco é possível detectar claramente os contornos.

Entretanto, devido a existência de ruídos, a função retorna uma grande quantidade de pontos identificados como contornos. Por isso, é necessário que ocorra uma filtragem na lista de contornos retornada, eliminando aqueles muito pequenos ou muito grandes e que não possuem formato quadrangular.

A lista de contornos resultante é então submetida à função *find_commands()*, que recebe também a imagem original e uma lista com as imagens base dos comandos.

Código 5 - Método find_commands()

```
70  def find_commands(cnts, image, train_commands):
71      if len(cnts) == 0:
72          return []
73
74      cnts = sort_contours(cnts, method="top-to-bottom")
75
76      cnts_order = []
77      line_cnts  = []
78      center, pts = define_center(cnts[0])
79      limitY = center[1] + LIMIT_Y_LINE
80      for i in range(len(cnts)):
81          center, pts = define_center(cnts[i])
82          if(center[1] < limitY):
83              line_cnts.append(cnts[i])
84          else:
85              limitY = center[1] + LIMIT_Y_LINE
86              line_cnts = sort_contours(line_cnts)
87              cnts_order.append(line_cnts)
88              line_cnts = []
89              line_cnts.append(cnts[i])
90      line_cnts = sort_contours(line_cnts)
91      cnts_order.append(line_cnts)
92
93      commands = []
94      for y in range(len(cnts_order)):
95          line_commands  = []
96          for x in range(len(cnts_order[y])):
97              qCommand = Query_command()
98              qCommand = preprocess_command(cnts_order[y][x],image)
99              #cv2.imwrite(str(x+y) +".jpeg", qCommand.command_img);
100             best_command_match, diff = match_command(qCommand,train_commands)
101             if(best_command_match != "Unknown"):
102                 qCommand.best_command_match, qCommand.diff = best_command_match, diff
103                 line_commands.append(qCommand)
104         check_line(line_commands, image)
105         if len(line_commands) > 0:
106             commands.append(line_commands)
107
108     return commands
```

Fonte: Os autores (2018)

A ordem em que as peças estão dispostas na imagem é de fundamental

importância para a aplicação, pois os comandos devem ser executados na ordem em que o usuário definiu. Como só é possível ordenar em um eixo por vez para saber a ordem dos contornos, esses são ordenados verticalmente em relação à imagem, para identificar a ordenação das linhas. Depois de agrupar quais comandos estão em cada linha, essas são submetidas a uma ordenação horizontal para saber a ordem das instruções correspondente à linha.

Após realizar a ordenação, todos os contornos são percorridos com o objetivo de reconhecer os comandos fotografados pelo usuário, comparando cada um dos contornos identificados com as imagens base de cada uma das instruções. Essa comparação entre os comandos é feita pela função *match_command()*, que calcula a diferença entre as duas imagens pelo método MSE (*Mean Squared Error*).

Código 6 - Método match_command()

```
135  def match_command(qCommand, train_command):
136      best_command_match_diff = 1000000000
137      best_command_match_name = "Unknown"
138      i = 0
139      retval, qCommand.command_img = cv2.threshold(qCommand.command_img, 100, 255, cv2. THRESH_BINARY)
140      if (len(qCommand.command_img) != 0):
141          for Tcommand in train_command:
142              err = np.sum((Tcommand.img.astype("float") - qCommand.command_img.astype("float")) ** 2)
143              err /= float(Tcommand.img.shape[0] * qCommand.command_img.shape[1])
144              if err < best_command_match_diff:
145                  best_command_match_diff = err
146                  best_command_name = Tcommand.name
147      if (best_command_match_diff < RANK_DIFF_MAX):
148          best_command_match_name = best_command_name
149      return best_command_match_name, best_command_match_diff
```

Fonte: Os autores (2018)

Primeiramente, as imagens são convertidas para valores em ponto flutuante e é calculada a diferença entre as duas elevando o resultado ao quadrado. O valor resultante é então dividido pelo número total de pixels nas imagens (ROSEBROCK, 2014). O valor resultante é o que define o quão similar são as imagens. Se o valor resultante for zero, as imagens são perfeitamente iguais, se for maior que isso, há alguma diferença (ROSEBROCK, 2014).

Como dificilmente o valor obtido será zero, foi definido um limite do quanto as imagens podem ser diferentes, mas ainda assim serem consideradas a mesma. Com base no valor resultante do MSE, verifica-se qual foi o melhor resultado obtido e define-se que aquele contorno específico corresponde à determinada instrução.

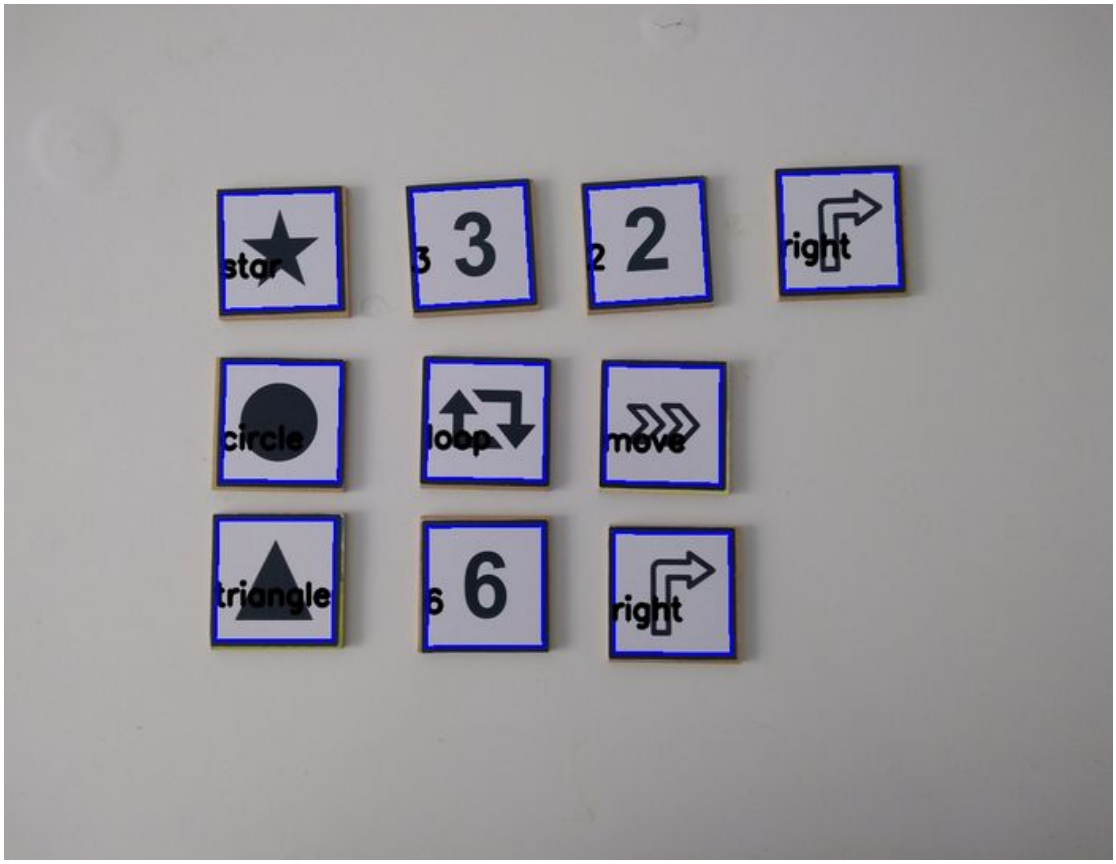
Para finalizar, no método *find_commands()* ainda é feito uma verificação nas linhas, para identificar contornos que se sobrepõem e removê-los. Isso é necessário porque o algoritmo de contornos por vezes reconhece dois contornos válidos em uma mesma peça. Essa verificação é feita pelo método *intersection()*.

Código 7 - Método intersection()

```
def intersection(cnt1, cnt2):
    leftmost_cnt1 = tuple(cnt1[cnt1[:,:,0].argmin()][0])
    rightmost_cnt1 = tuple(cnt1[cnt1[:,:,0].argmax()][0])
    topmost_cnt1 = tuple(cnt1[cnt1[:,:,1].argmin()][0])
    bottommost_cnt1 = tuple(cnt1[cnt1[:,:,1].argmax()][0])

    leftmost_cnt2 = tuple(cnt2[cnt2[:,:,0].argmin()][0])
    rightmost_cnt2 = tuple(cnt2[cnt2[:,:,0].argmax()][0])
    topmost_cnt2 = tuple(cnt2[cnt2[:,:,1].argmin()][0])
    bottommost_cnt2 = tuple(cnt2[cnt2[:,:,1].argmax()][0])

    if leftmost_cnt1[0] < leftmost_cnt2[0] and rightmost_cnt1[0] > leftmost_cnt2[0]:
        return True
    else:
        return False
```

Fonte: Os autores (2018)

Figura 26 - Resultado expresso em imagem

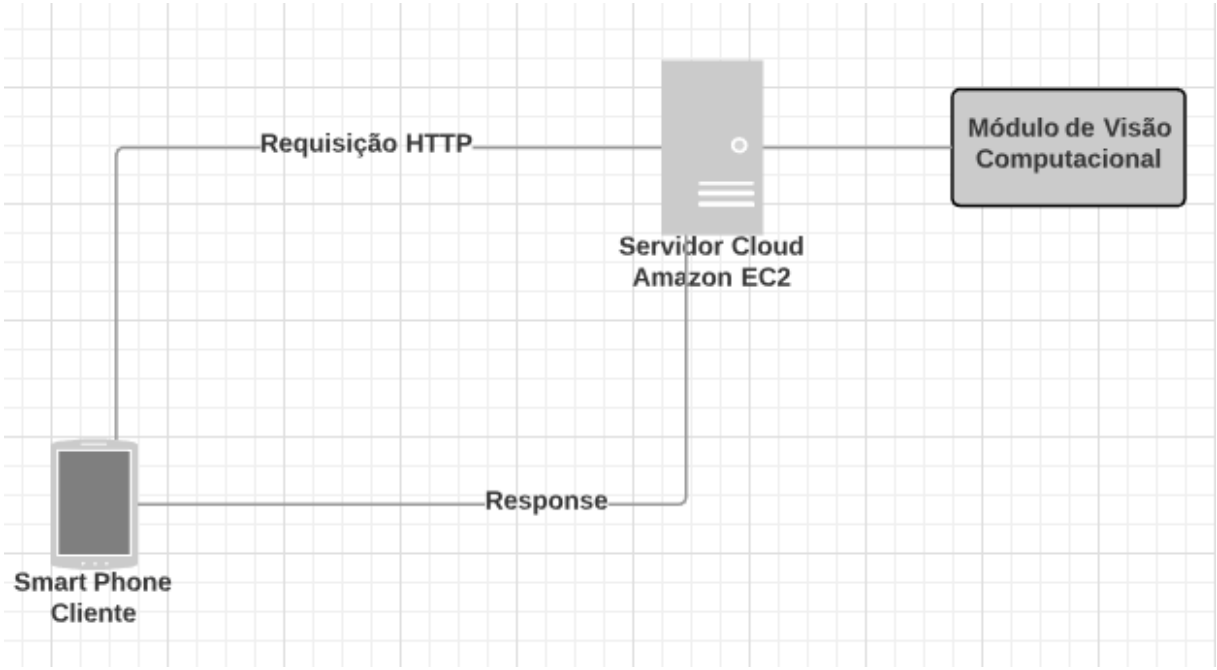


Fonte: Os autores (2018)

6.4.3 **INTEGRAÇÃO ENTRE JOGO E RECONHECIMENTO DE IMAGEM.**

Após desenvolver o aplicativo e *script* de Visão Computacional para reconhecer as peças, foi necessário encontrar formas de integrar ambos. Como a ferramenta OpenCV e Unity não compartilham linguagem de programação em comum, limitando assim o uso conjunto das duas ferramentas, a integração dos módulos foi feita por meio de uma aplicação cliente/servidor, exemplificada no diagrama 2.

Diagrama 2 - Estrutura da aplicação



Fonte: Os autores (2018)

A integração entre os módulos foi feita visando o baixo acoplamento entre os dois, com o objetivo de que eles funcionem de forma independente e que alterações em um, não prejudiquem o funcionamento do outro. Foi buscado, na parte do aplicativo, deixar a funcionalidade de reconhecimento dos comandos isolada, assim, caso seja necessário mudar o método de reconhecimento para outra aplicação ou algoritmo, não haverá um impacto muito grande dentro do código do jogo, permitindo assim futuramente alterar a maneira de reconhecer os comandos para dentro do celular, otimizando a aplicação e retirando a restrição de estar conectado na internet.

Para integrar os módulos a aplicação realiza uma requisição para o servidor, contendo a imagem que deseja-se identificar os comandos. Essa imagem deve ter sido antes convertida ao formato de Base64. O servidor recebe essa requisição, e aplica o algoritmo de reconhecimento dos comandos, retornando para o cliente uma string com os comandos reconhecidos na imagem enviada.

O servidor contém uma aplicação que foi escrita na linguagem Python, a mesma que os scripts de Visão Computacional e utilizando o *framework web* Flask. O Flask é um *framework* minimalista, de tamanho reduzido e com pouca ou nenhuma dependência de bibliotecas externas, *open source* e escrito na linguagem Python (KUSHAL DAS). Esta ferramenta foi utilizada neste projeto para lidar com as requisições http (emitidas pelo aplicativo do *smartphone*) e também como servidor da aplicação, sendo portanto utilizado com um servidor *web*.

Este módulo foi hospedado em uma máquina virtual na modalidade EC2 da Amazon Web Services, com o Sistema Operacional Linux AMI 2018.03.0.

O trecho de código 8 contém o algoritmo escrito no arquivo App.py, que contém a implementação da aplicação Flask (que faz a interface com o OpenCV, também em execução na mesma máquina virtual).

Código 8 - Código do servidor Flask

```
#!flask/bin/python
from flask import Flask
from flask import request
import numpy as np
import Command
import cv2

app = Flask("Card-Detector")

@app.route('/', methods=['POST'])
def post():
    content = request.get_json()
    return getCommandByImage(content['image'])

def readb64(base64_string):
       nparr = np.fromstring(base64_string.decode('base64'), np.uint8)
       img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
       return img

def getCommandByImage(content):
       image = readb64(content)
       r = 1100.0 / image.shape[1]
       dim = (1100, int(image.shape[0] * r))
       image = cv2.resize(image,dim, interpolation = cv2.INTER_AREA)
       pre_proc = Command.preprocess_image(image)
       cnts, qntd_found, qtnd_squard = Command.find_cnts_commands(pre_proc)
       commands = Command.find_commands(cnts, image)
       return Command.responseCommands(commands)

app.run(host='0.0.0.0', port=80)
```

Fonte: Os autores (2018)

Ao receber uma requisição do tipo *Post* para o endereço raiz do servidor, o

framework Flask encaminha a requisição para o método *post()*, que recebe a imagem e envia para o método *getCommandByImage()*. Esse método ajusta as dimensões da imagem e faz interações com o script "*Command.py*", cujo funcionamento já foi descrito na seção 6.4.2 deste trabalho e que descreve as instruções identificadas em um texto no formato JSON que é enviado como resposta para o aplicativo.

Do ponto de vista do cliente, quem realiza a ação de integração é o *script* "*RequestManager*". O trecho de código 9 mostra o algoritmo utilizado para enviar a fotografia captada pelo usuário no celular para o servidor.

Código 9 - Script RequestManager

```
public class RequestManager{

    public static IEnumerator Request(byte[] bytes, RecognizeCommandManager recognizeCommandManager)
    {
        string json = getJsonRequest(getImage64(bytes));
        UnityWebRequest www = UnityWebRequest.Post("http://ec2-18-224-2-172.us-east-2.compute.amazonaws.com", json);
        byte[] bytesRequest = Encoding.UTF8.GetBytes(json);
        UploadHandlerRaw uH = new UploadHandlerRaw(bytesRequest);
        uH.contentType = "application/json";
        www.uploadHandler = uH;
        {
            yield return null;
            yield return www.SendWebRequest();
            if (www.isNetworkError)
            {
                recognizeCommandManager.response(Messages.PROBLEMA_CONEXAO, true);
            }
            else if(www.isHttpError || www.responseCode != 200){
                recognizeCommandManager.response(Messages.ERRO_SERVIDOR + www.responseCode + " " + www.error  ,  true);
            }
            else
            {
                recognizeCommandManager.response(www.downloadHandler.text, false);
            }
        }
    }
```

Fonte: Os autores (2018)

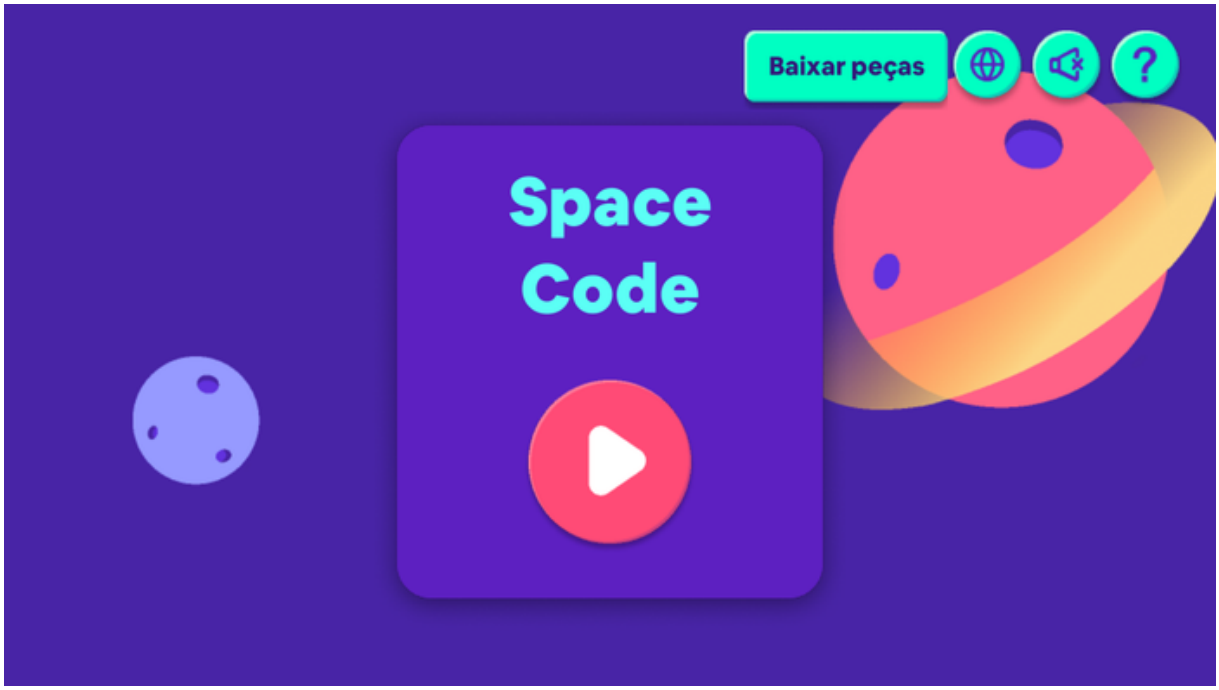
A imagem é primeiro convertida para um *string* de caracteres em base 64 e anexada a um JSON, que é então enviado para o endereço de DNS público fornecido pela Amazon e espera pela resposta da requisição, retornando-a para o script "*RecognizeCommandManager*" que dá continuidade ao fluxo da aplicação, mostrando para o usuário se ocorreu um erro ou quais foram as peças identificadas na fotografia.

7 **RESULTADOS**

Ao final do processo de desenvolvimento, foram finalizados com sucesso tanto o jogo quanto o servidor *web* contendo o módulo de Visão Computacional. Para que o jogo funcionasse em dispositivos *mobile* da plataforma Android, o mesmo foi exportado, através do Unity Editor, como um arquivo em formato APK que é a extensão instalável no sistema operacional Android.

O arquivo gerado foi testado em 15 dispositivos com o objetivo de verificar como o aplicativo se comportava em aparelhos de diferentes capacidades e resoluções, obtendo bons resultados. As figuras 27,28 e 29 mostram o visual final das três cenas do jogo.

Figura 27 - Tela inicial - versão final



Fonte: Os autores (2018)

Figura 28 - Tela de seleção de fases - versão final

Fonte: Os autores (2018)

Figura 29 - Tela fase - versão final

Fonte: Os autores (2018)

Foram criadas 20 fases diferentes, abordando diversos conceitos de

Pensamento Computacional, como exemplificado na seção 5.3. Todos os requisitos funcionais e não funcionais foram implementados.

O jogo possui algumas limitações, como por exemplo o fato de que o dispositivo *mobile* precisa estar conectado à internet para que possa se comunicar com o servidor que contém o módulo de Visão Computacional. Uma outra limitação é que o usuário deve manter o dispositivo em posição estática ao capturar a foto, pelo menos até que a mensagem “Carregando” apareça na tela, para conseguir resultados dos comandos mais precisos. Também é necessário que o aparelho possua uma câmera fotográfica traseira funcional.

As peças devem seguir o modelos dos Apêndice A e B. Nestes modelos, as peças tem um tamanho de 5cm x 5cm quando impressas em folha A4. Alterações nos símbolos definidos fará com que as peças não sejam reconhecidas pelo módulo de Visão Computacional. As peças também devem ter contornos fortes para que os algoritmos possam identificá-las normalmente. As cores das peças não devem ser alteradas.

Foram realizados também alguns testes sobre o módulo de Visão Computacional a fim de garantir sua eficiência e identificar os melhores cenários e limites do sistema. Para isso foram impressas peças em papel adesivo que coladas em peças de madeira. Foi criado um banco de imagens com cerca de 209 imagens subdivididas em 13 categorias:

- **Peças tortas:** Fotografias onde as peças dos comandos estão tortas;
- **Peças mal recortadas:** Fotografias onde as peças dos comandos estão com algum pedaço faltando, com a borda mal recortada;
- **Peças amassadas:** Fotografias onde as peças dos comandos estão amassadas;
- **Linhas tortas:** Fotografias em que comandos pertencentes a mesma função estão desalinhados;
- **Fundos diferentes:** Fotografias onde as peças dos comandos estão sobre fundos de cores ou texturas diferentes;
- **Distância:** Fotografias com diferentes distâncias entre a câmera e as peças de comandos;
- **Ângulo:** Fotografias com diferentes ângulos entre a câmera e as peças de comandos;
- **Ideal:** Fotografias que obedecem um cenário ideal de reconhecimento;
- **Luz:** Fotografias que há um ponto de luz muito forte incidindo sobre as peças, gerando reflexos em alguns pontos.
- **Sombra:** Fotos em que incide sombra sobre as peças.
- **Nenhuma peça:** Fotos em que não consta nenhuma peça. Categoria

criada para verificar como o servidor reage a este tipo de imagem buscando a identificação de possíveis erros.

- **Luz do dia:** Fotos capturadas em iluminação natural ao ar livre.
- **Papel:** Fotos em que as peças utilizadas foram impressas em papel simples e com menor qualidade.

Foi criado também um *script* escrito em Python que aplicava os algoritmos de Visão Computacional desenvolvidos em cada uma das imagens, comparava o resultado obtido com o resultado esperado, descrito em um arquivo de texto pelos autores, e escrevia em um outro arquivo o resultado da comparação e o resultado obtido. O quadro 3 apresenta os resultados dos testes, por categoria.

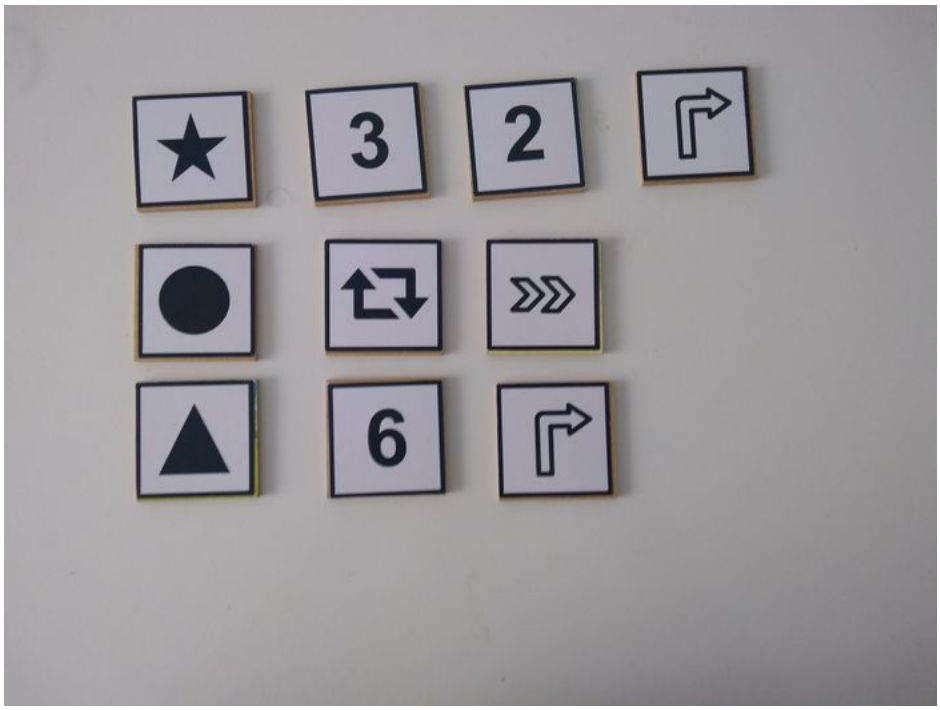
Quadro 3 - Testes do módulo de Visão Computacional

Categoria	Quantidade de imagens	Imagens corretamente reconhecidas
Ideal	32	32
Amassadas	4	0
Ângulo	22	19
Distância	13	5
Fundos diferentes	22	12
Linhas tortas	9	7
Luz	23	7
Mal recortadas	7	0
Peças tortas	16	16
Sombra	14	12
Nenhuma peça	14	11
Papel	12	12
Luz do Dia	21	11

Fonte: Os autores (2018)

No geral, o módulo se saiu bem ao identificar as peças nas imagens, tendo como cenário ideal uma foto tirada em paralelo às peças, em um ambiente bem iluminado, com pouca ou nenhuma sombra incidindo sobre as peças e com as mesmas dispostas em um fundo branco, como pode ser visto na figura 30. A categoria “Ideal” contém fotografias nestas condições.

Figura 30 - Exemplo de fotografia no cenário ideal



Fonte: Os autores (2018)

Foram encontradas algumas limitações, não sendo possível reconhecer as peças nas seguintes situações:

- Fotos com forte incidência de sombras sobre as imagens;
- Fotos em que as peças estão dispostas sobre fundo preto (ou muito escuro), ou sobre fundos que não sejam lisos e que possuem alguma forma geométrica no relevo;
- Fotos onde um ponto de luz gera reflexo sobre as imagens, situação que pode acontecer dependendo do material utilizado nas peças ou da superfície onde as peças estão.
- Fotos onde as peças estão mal recortadas (com pedaços das bordas ausentes),muito amassadas (considerando as peças feitas em papel), ou aquelas em que a pintura está fraca ou com falhas;
- Fotos capturadas muito distante das peças (resultando em imagens dos comandos muito pequenas para serem identificados).
- Fotos em que as peças estão muito próximas umas das outras, pode fazer com que algumas instruções não sejam reconhecidas.
- Fotos em que as peças ficam para fora da imagem;

O ângulo em que a foto é tirada não proporcionou grandes alterações no reconhecimento das peças, embora ângulos muito agudos devam ser evitados.

Todo o código desenvolvido neste projeto, relativo tanto ao jogo quanto ao

módulo de Visão Computacional, foram armazenados em repositórios públicos na plataforma de versionamento de código GitHub com os nomes de SpaceCode e SpaceCode - Reconhecimento(COELHO, 2018).

8 **CONSIDERAÇÕES FINAIS**

O objetivo deste trabalho de conclusão de curso foi criar um jogo executável na plataforma *mobile* Android que tem como função servir como ferramenta de apoio ao desenvolvimento de Pensamento Computacional em crianças. Além disso, foi proposto que fossem utilizadas Interfaces Tangíveis como forma de interação entre usuários e aplicativo.

Para cumprir esses objetivos, foi iniciado um estudo sobre os principais assuntos relacionados ao projeto com o propósito de compreender melhor os conceitos de Pensamento Computacional e Interfaces Tangíveis, bem como qual a influência esses conceitos exercem no processo de aprendizado e sobre as capacidades cognitivas dos alunos.

Em seguida, foi analisado o estado da arte referente aos jogos voltados para o desenvolvimento de Pensamento Computacional e ao uso de Interfaces Tangíveis em *softwares* deste tipo. Essa etapa serviu para aprimorar a ideia do *software* a ser desenvolvido e analisar qual seria a melhor abordagem para o projeto.

A fase de desenvolvimento foi repleta de desafios. Tanto a produção do aplicativo utilizando o Unity quanto o desenvolvimento dos algoritmos de Visão Computacional utilizando o OpenCV exigiram uma grande curva de aprendizado para que houvesse sucesso na implementação, mas ambas supriram completamente as necessidades do projeto. Escolher ferramentas bastante utilizadas foi de fundamental importância, pois em diversos momentos foi necessário recorrer a fóruns de desenvolvimento para esclarecer dúvidas ou buscar ideias.

Por fim, todos os artefatos necessários para o sucesso deste trabalho foram desenvolvidos e funcionam de forma satisfatória. Dentre os objetivos traçados na definição da proposta do jogo, é possível dizer que se obteve sucesso ao criar um jogo com peças de produção baratas e simples, ideal para ambientes escolares, e que foi possível adicionar uma grande mobilidade ao dispositivo mobile, uma vez que esse não precisa ficar em uma posição fixa.

Analisando as características desejáveis em aplicações que utilizam Interfaces Tangíveis voltadas para a educação, definidas por FALCÃO e GOMES (2007) e descritas na seção 2.2.1, pode-se dizer que esta aplicação atende à algumas delas:

- **Independência do computador pessoal:** Um único celular pode facilmente ser utilizado para avaliar a solução de diversos usuários que podem interagir apenas com as peças.
- **Uso colaborativo:** os usuários podem formar grupos para pensar na solução do problema, uma vez que podem utilizar exclusivamente as peças

para montar as soluções.

- **Scaffolding e diferentes níveis de dificuldade:** as fases elaboradas possuem um nível progressivo de dificuldade e também é possível definir mensagens de ajuda específicas para cada um dos níveis.

Não é possível afirmar que este trabalho se adequa aos conceitos de simplicidade e engajamento, uma vez que nenhuma análise referente à estas questões foi feita.

A finalização deste projeto abre portas para diversas melhorias e trabalhos futuros. No aplicativo podem ser implementadas novas traduções para outros idiomas, ampliando a questão da internacionalização do *software.*

No módulo de Visão Computacional é possível que a utilização de métodos de *Deep Learning*, bem como melhorias no processamento de imagem, principalmente no tratamento de sombras, possa gerar melhorias significativas no reconhecimento correto das instruções.

Também é possível tentar transferir este módulo para dentro do dispositivo *mobile*, evitando assim que o aplicativo fique dependente de conexão com a internet para funcionar. Novos tipos de instruções podem ser inseridos no sistema, ampliando a variedade de desafios possíveis e aumentando a gama de conceitos abordados nas fases.

A especificação das fases em arquivos no formato JSON possibilita que novos níveis sejam facilmente criados, o que abre espaço para diversas evoluções deste trabalho. É possível explorar essa funcionalidade para criar fases específicas para algum curso ou aula, criar fases com outras dificuldades abordando novos conceitos e problemáticas, aumentando a faixa etária alvo do jogo, e permitir que os próprios jogadores criem as suas fases (aumentando assim o engajamento e estimulando a criatividade) e desafiem outros usuários.

Por fim, devido à limitações de tempo e de escopo deste trabalho, não foi possível fazer uma análise da efetividade do uso desta aplicação como ferramenta de apoio ao desenvolvimento de Pensamento Computacional. Esta análise poderá ser feita futuramente, em um projeto piloto, definindo um conteúdo programático a ser abordado tendo o aplicativo como auxílio e analisando os resultados obtidos.

REFERÊNCIAS

BARR, Valerie; STEPHENSON, Chris. Bringing computational thinking to K-12: what is Involved and what is the role of the computer science education community?. **Acm Inroads**, v. 2, n. 1, p. 48-54, 2011.

BLIKSTEIN , Paulo. **O pensamento computacional e a reinvenção do computador na educação**. 2008. Disponível em: <http://www.blikstein.com/paulo/documents/online/ol_pensamento_computacional.html>. Acesso em: 10 nov. 2017.

BUENO, Fabrício. Jogo Educacional para o Ensino de Estatística. In: SBGAMES 2010. 2010. **Anais...**. 2010. 253-256 p.

CHRISTOPOULOU, Eleftheria; XINOGALOS, Stelios. Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. **International Journal of Serious Games**, v. 4, n. 4, p. 21-36, 2017.

CODE.ORG. **Artista**. Disponível em: <https://studio.code.org/s/artist/stage/1/puzzle/1>. Acesso em: 15 nov. 2017.

COELHO, Everton S.. **Repositórios do usuário evertonscoelho**. 2018. Disponível em: <https://github.com/evertonscoelho?tab=repositories>. Acesso em: 23 nov. 2018.

CSTA. **Computational thinking teacher resources**: Second Edition. 2011. Disponível em: <https://c.ymcdn.com/sites/www.csteachers.org/resource/resmgr/472.11CTTeacherResources_2ed.pdf>. Acesso em: 15 nov. 2017.

DOS REIS, Alessandro Vieira; GONÇALVES, Berenice dos Santons. Interfaces Tangíveis: Conceituação e Avaliação. **Estudos em Design**, v. 24, n. 2, 2016.

EDJEELETRONICS. OpenCV-Playing-Card-Detector. **GitHub**. 2017. Disponível em: <https://github.com/EdjeElectronics/OpenCV-Playing-Card-Detector>. Acesso em: 1 jul. 2018.

ELHADY, Hady. Top Game engines in 2018. **InstaBug**. 2017. Disponível em: <https://instabug.com/blog/game-engines/>. Acesso em: 27 out. 2018.

FALCÃO, Taciana Pontual; GOMES, Alex Sandro. Interfaces tangíveis para a educação. In: BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMÁTICA NA EDUCAÇÃO-SBIE). 2007. 579-589 p.

FGV. **28ª Pesquisa Anual do Uso de TI 2017**. 2017. Disponível em: <https://eaesp.fgv.br/ensinoeconhecimento/centros/cia/pesquisa>. Acesso em: 10 nov. 2017.

FISHKIN, Kenneth P. . A taxonomy for and analysis of tangible interfaces. **Personal and Ubiquitous Computing**, v. 8, n. 5, p. 347-358, 2004.

FLORIAN. The best Game Engines for beginners. **Website Tool Tester**. 2018. Disponível em: <>. Acesso em: 27 out. 2018.

GOOGLE CLOUD. **Cloud Vision**. Disponível em: <https://cloud.google.com/vision/>. Acesso em: 4 out. 2018.

GOOGLE DEVELOPERS. **Blockly**. Disponível em: <https://developers.google.com/blockly/>. Acesso em: 15 nov. 2017.

IBM. **Watson**. Disponível em: <https://www.ibm.com/watson/>. Acesso em: 4 out. 2018.

IDC. Smartphone OS Market Share. **IDC Analysis the future**. 2016. Disponível em: <https://www.idc.com/promo/smartphone-market-share/os>. Acesso em: 5 dez. 2018.

KANAREK BRUNEL, Guilherme. **Pagina Pessoal do Linkedin**. 2018. Disponível em: <https://www.linkedin.com/in/guilhermekanarek/>. Acesso em: 20 nov. 2018.

KLEIMAN, Glen M. Myths and realities about technology in K–12 schools. **LNT Perspectives**, n. 14, 2000.

KUSHAL DAS. Introduction to Flask. **Pymbook**. Disponível em: <https://pymbook.readthedocs.io/en/latest/flask.html >. Acesso em: 20 set. 2018.

LEARNING RESOURCES. **Robot Mouse Coding Activity Set**: Activity Set. Disponível em: <https://vidweb.aws.marketlive.com/learningresources_vid/text/pdf/LER2831_guide.pdf>. Acesso em: 16 nov. 2017.

LIGHTBOT INC. **Lightbot**. Disponível em: <<http://lightbot.com/>. Acesso em: 16 nov. 2017.

MAGE STUDIO - KID GAME. **Robotizen**. Disponível em: <https://play.google.com/store/apps/details?id=vn.com.mage.irobot&hl=en>. Acesso em: 16 nov. 2017.

MARENGONI, Maurício; STRINGHINI, Denise. Tutorial: Introdução à Visão Computacional usando OpenCV. **Revista de Informática Teórica e Aplicada**, v. 16, n. 1, p. 125 - 160, 2009.

MARSHALL, Paul. Do tangible interfaces enhance learning?. In: PROCEEDINGS OF THE 1ST INTERNATIONAL CONFERENCE ON TANGIBLE AND EMBEDDED INTERACTION.. 2007. ACM, 2007. 167-170 p.

MATTEL. **Code-a-Pillar**. Disponível em: <http://fisher-price.mattel.com/shop/en-us/fp/think-learn/think-learn-code-a-pillar-starter-gift-set-fgn83>. Acesso em: 16 nov. 2017.

MEIRELLES, Fernando S. 28ª Pesquisa Anual do Uso de TI 2017. **FGV EAESP**. 2017. Disponível em: <http://eaesp.fgvsp.br/sites/eaesp.fgvsp.br/files/pesti2017gvciappt.pdf>. Acesso em: 10 nov. 2017.

MIT MEDIA LAB, Lifelong Kindergarten. **Scratch**. Disponível em: <https://scratch.mit.edu/>. Acesso em: 15 nov. 2017.

MOYNIHAN, Tim. **Osmo Turns Blocks Into Code to Teach Kids Programming**. 2016. Disponível em: <https://www.wired.com/2016/05/osmo-turns-blocks-code-teach-kids-programming>. Acesso em: 19 nov. 2017.

OPENCV TEAM. **About**. Disponível em: <https://opencv.org/about.html>. Acesso em: 2 set. 2018.

OSMO. **Osmo Coding**. Disponível em: <https://www.playosmo.com/en>. Acesso em: 17 nov. 2018.

PDM INC. **Cubico**. Disponível em: <http://www.cubicoding.com/sub/cubico_games.php>. Acesso em: 17 nov. 2017.

PIAGET, Jean. How children form mathematical concepts. **Scientific Ameriacan**, v. 189, n. 5, p. 74-79, 1953.

PRIMO TOYS. **Cubeto**. Disponível em: <https://www.primotoys.com>. Acesso em: 16 nov. 2017.

ROSEBROCK, Adrian. How-To: Python Compare Two Images. **Pyimagesearch**. 2014. Disponível em: <https://www.pyimagesearch.com/2014/09/15/python-compare-two-images/>. Acesso em: 9 set. 2018.

SPACE CODE. **Space Code**. 2018. Disponível em: <https://play.google.com/store/apps/details?id=com.EvertonYuri.TCC>. Acesso em: 5 dez. 2018.

TECHTERMS. Sprite. **TechTerms**. 2012. Disponível em: <https://techterms.com/definition/sprite>. Acesso em: 17 out. 2018.

TENSOR FLOW. **Tensor Flow**. Disponível em: <https://www.tensorflow.org/>. Acesso em: 4 out. 2018.

UNITY TECHNOLOGIES. **2D UFO Tutorial**. Disponível em: <https://unity3d.com/pt/learn/tutorials/s/2d-ufo-tutorial R>. Acesso em: 15 mar. 2018.

_____. **Game Objects**. Disponível em: <https://docs.unity3d.com/Manual/class-GameObject.html >. Acesso em: 17 out. 2018.

_____. **Prefabs**. Disponível em: <https://docs.unity3d.com/Manual/Prefabs.html>. Acesso em: 15 out. 2018.

_____. Relações Públicas. **Unity 3d**. Disponível em: <https://unity3d.com/pt/public-relations>. Acesso em: 17 jul. 2018.

UNITY TECHNOLOGIES. **Scenes**. Disponível em: <https://docs.unity3d.com/Manual/CreatingScenes.html>. Acesso em: 5 dez. 2018.

UNITY TECHNOLOGIES. **Sprites**. Disponível em: <

https://docs.unity3d.com/Manual/Sprites.html >. Acesso em: 17 out. 2018.

VON WANGENHEIM, Christiane Gresse; NUNES, Vinícius Rodrigues; DOS SANTOS, Giovane Daniel. Ensino de computação com scratch no ensino fundamental–um estudo de caso. **Revista Brasileira de Informática na Educação**, v. 22, n. 03, p. 115, 2014.

WIKIPEDIA. **List of game engines**. Disponível em: <https://en.wikipedia.org/wiki/List_of_game_engines>. Acesso em: 8 fev. 2018.

WING, Jeannette M. Computational thinking. **Communications of the ACM**. Nova York -NY, v. 49, n. 3, p. 33-35, 2006.

WING, Jeannete M . Computational thinking benefits society. **SOCIAL ISSUES IN COMPUTING**. 2014. Disponível em: <http://socialissues.cs.toronto.edu/index.html?p=279.html>. Acesso em: 10 nov. 2017.

WIZBICKI, Andreia Schimanowski; BATTISTI, Gerson. RECONHECIMENTO DE PADRÕES EM IMAGENS APLICANDO VISÃO COMPUTACIONAL. In: SEMINÁRIO DE INICIAÇÃO CIENTÍFICA, XXII. 2014.

YAROSLAVSKI, Danny. **How does Lightbot teach programming?**. 2014. Disponível em: <http://www.clevertouch.com/wp-content/uploads/2016/04/Lightbot_HowDoesLightbotTeachProgramming.pdf>. Acesso em: 22 nov. 2017.

APÊNDICE A — CÓDIGO VISAO COMPUTACIONAL

```
import numpy as np
import cv2
import time
import os

### Constants ###

# Adaptive threshold levels
CARD_THRESH = 30

# Dimensions of rank train images
COMMAND_WIDTH = 168
COMMAND_HEIGHT = 148

RANK_DIFF_MAX = 20000

COMMAND_MAX_AREA = 120000
COMMAND_MIN_AREA = 5000

LIMIT_Y_LINE = 80

font = cv2.FONT_HERSHEY_SIMPLEX

path = os.path.dirname(os.path.abspath(__file__))

class Query_command:
    def __init__(self):
        self.contour = []
        self.width, self.height, self.x, self.y = 0, 0, 0, 0
        self.rect = []
        self.center = []
        self.warp = []
        self.command_img = []
        self.best_command_match = "Unknown"
        self.diff = 0
```

```
class Train_command:
    def __init__(self):
        self.imgs = []
        self.name = "Placeholder"

def load_commands(filepath):
    train_commands = []
    i = 0
    for Command in ['2','3','4','5','6','7','8','9','circle','left','right']:
        imgs = []
        train_commands.append(Train_command())
        train_commands[i].name = Command
        filename = Command + '.jpg'
        imgs.append(cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE))
        train_commands[i].img = imgs
        i = i + 1

    for Command in ['loop','loop2']:
        imgs = []
        train_commands.append(Train_command())
        train_commands[i].name = 'loop'
        filename = Command + '.jpg'
        imgs.append(cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE))
        train_commands[i].img = imgs
        i = i + 1

    for Command in ['move', 'move2','move3','move4']:
        imgs = []
        train_commands.append(Train_command())
        train_commands[i].name = 'move'
        filename = Command + '.jpg'
        imgs.append(cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE))
        train_commands[i].img = imgs
        i = i + 1

    for Command in ['star', 'star2','star3','star4']:
        imgs = []
        train_commands.append(Train_command())
        train_commands[i].name = 'star'
```

```
            filename = Command + '.jpg'
            imgs.append(cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE))
            train_commands[i].img = imgs
            i = i + 1

        for Command in ['triangle', 'triangle2','triangle3','triangle4']:
            imgs = []
            train_commands.append(Train_command())
            train_commands[i].name = 'triangle'
            filename = Command + '.jpg'
            imgs.append(cv2.imread(filepath+filename, cv2.IMREAD_GRAYSCALE))
            train_commands[i].img = imgs
            i = i + 1

        return train_commands

    train_commands = load_commands( path + '/Commands_Imgs/')

    def preprocess_image(image):
        gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray,(5,5),0)
                                                    thresh =
cv2.adaptiveThreshold(blur,255,cv2.ADAPTIVE_THRESH_MEAN_C,
cv2.THRESH_BINARY,11,2)
        return thresh

    def find_cnts_commands(thresh_image):
                                                dummy,cnts,hier =
cv2.findContours(thresh_image,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
        cnts_return = []
        for i in range(len(cnts)):
            size = cv2.contourArea(cnts[i])
            peri = cv2.arcLength(cnts[i],True)
            approx = cv2.approxPolyDP(cnts[i],0.01*peri,True)
              if len(approx) == 4 and (size < COMMAND_MAX_AREA) and (size >
COMMAND_MIN_AREA):
                #print(size)
                cnts_return.append(cnts[i])
```

```
    return cnts_return, len(cnts), len(cnts_return)

    def find_commands(cnts, image):
        if len(cnts) == 0:
            return []

        cnts = sort_contours(cnts, method="top-to-bottom")

        cnts_order = []
        line_cnts = []
        center, pts = define_center(cnts[0])
        limitY = center[1] + LIMIT_Y_LINE
        for i in range(len(cnts)):
            center, pts = define_center(cnts[i])
            if(center[1] < limitY):
                line_cnts.append(cnts[i])
            else:
                limitY = center[1] + LIMIT_Y_LINE
                line_cnts = sort_contours(line_cnts)
                cnts_order.append(line_cnts)
                line_cnts = []
                line_cnts.append(cnts[i])
        line_cnts = sort_contours(line_cnts)
        cnts_order.append(line_cnts)

        commands = []
        for y in range(len(cnts_order)):
            line_commands = []
            for x in range(len(cnts_order[y])):
                qCommand = Query_command()
                qCommand = preprocess_command(cnts_order[y][x],image)
                                                    best_command_match,     diff     =
match_command(qCommand,train_commands)
                if(best_command_match != "Unknown"):
                                    qCommand.best_command_match,  qCommand.diff  =
best_command_match, diff
                    line_commands.append(qCommand)
            check_line(line_commands, image)
            if len(line_commands) > 0:
```

```
                commands.append(line_commands)

            return commands

        def check_line(line_commands, img):
            for x in range(len(line_commands)):
                for y in range(x+1, len(line_commands)):
                    if intersection(line_commands[x].contour, line_commands[y].contour):
                        line_commands.remove(line_commands[x])
                        break

        def intersection(cnt1, cnt2):
            leftmost_cnt1 = tuple(cnt1[cnt1[:,:,0].argmin()][0])
            rightmost_cnt1 = tuple(cnt1[cnt1[:,:,0].argmax()][0])
            topmost_cnt1 = tuple(cnt1[cnt1[:,:,1].argmin()][0])
            bottommost_cnt1 = tuple(cnt1[cnt1[:,:,1].argmax()][0])

            leftmost_cnt2 = tuple(cnt2[cnt2[:,:,0].argmin()][0])
            rightmost_cnt2 = tuple(cnt2[cnt2[:,:,0].argmax()][0])
            topmost_cnt2 = tuple(cnt2[cnt2[:,:,1].argmin()][0])
            bottommost_cnt2 = tuple(cnt2[cnt2[:,:,1].argmax()][0])

                if leftmost_cnt1[0] < leftmost_cnt2[0] and rightmost_cnt1[0] >
leftmost_cnt2[0]:
                return True
            else:
                return False


        def match_command(qCommand, train_command):
            best_command_match_diff = 1000000000
            best_command_match_name = "Unknown"
            best_command_name = "Unknown"
            i = 0
                                        retval,          qCommand.command_img          =
cv2.threshold(qCommand.command_img, 100, 255, cv2. THRESH_BINARY)
            if (len(qCommand.command_img) != 0):
                for Tcommand in train_command:
```

```
        for img in Tcommand.img:
                                                err = np.sum((img.astype("float") -
qCommand.command_img.astype("float")) ** 2)
                    err /= float(img.shape[0] * img.shape[1])
                    if err < best_command_match_diff:
                        best_command_match_diff = err
                        best_command_name = Tcommand.name
            if (best_command_match_diff < RANK_DIFF_MAX):
                best_command_match_name = best_command_name
            return best_command_match_name, best_command_match_diff

        def responseCommands(commands):
            response = ""
            if len(commands) == 0:
                return "Unknown"
            for y in range(len(commands)):
                for x in range(len(commands[y])):
                    if x != len(commands[y])-1:
                        response = response + commands[y][x].best_command_match +","
                    else:
                        response = response + commands[y][x].best_command_match
                if y != len(commands)-1:
                    response = response + ",NEXT,"
            return response


        def preprocess_command(contour, image):
            qCommand = Query_command()
            qCommand.contour = contour
                  qCommand.width, qCommand.height,qCommand.x, qCommand.y =
cv2.boundingRect(contour)
                qCommand.rect = (qCommand.width, qCommand.height,qCommand.x,
qCommand.y)
            qCommand.center, pts = define_center(contour)
                 qCommand.command_img = flattener(image, pts, qCommand.width,
qCommand.height)
            return qCommand

        def define_center(contour):
```

```
            peri = cv2.arcLength(contour,True)
            approx = cv2.approxPolyDP(contour,0.01*peri,True)
            pts = np.float32(approx)
            average = np.sum(pts, axis=0)/len(pts)
            cent_x = int(average[0][0])
            cent_y = int(average[0][1])
            return [cent_x, cent_y], pts

        def sort_contours(cnts, method="left-to-right"):
            i = 0
            if method == "top-to-bottom":
                i = 1
            boundingBoxes = [cv2.boundingRect(c) for c in cnts]
            (cnts, boundingBoxes) = zip(*sorted(zip(cnts, boundingBoxes), key=lambda
b:b[1][i], reverse=False))
            return cnts

        def flattener(image, pts, w, h):
            temp_rect = np.zeros((4,2), dtype = "float32")
            s = np.sum(pts, axis = 2)
            tl = pts[np.argmin(s)]
            br = pts[np.argmax(s)]
            diff = np.diff(pts, axis = -1)
            tr = pts[np.argmin(diff)]
            bl = pts[np.argmax(diff)]

            temp_rect[0] = tl
            temp_rect[1] = tr
            temp_rect[2] = br
            temp_rect[3] = bl

            maxWidth = 185
            maxHeight = 185
                    dst = np.array([[0,0],[maxWidth-1,0],[maxWidth-1,maxHeight-1],[0,
maxHeight-1]], np.float32)

            M = cv2.getPerspectiveTransform(temp_rect,dst)
            warp = cv2.warpPerspective(image, M, (maxWidth, maxHeight))
```

```
        warp = cv2.cvtColor(warp,cv2.COLOR_BGR2GRAY)
           return warp

        #!flask/bin/python
        from flask import Flask
        from flask import request
        from flask import send_file
        import numpy as np
        import Command
        import cv2

        app = Flask("Card-Detector")

        @app.route('/', methods=['POST'])
        def post():
           content = request.get_json()
           return getCommandByImage(content['image'])

        def readb64(base64_string):
         nparr = np.fromstring(base64_string.decode('base64'), np.uint8)
         img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
         return img

        @app.route("/pecas/")
        def DownloadPecas ():
                              return    send_file('/home/ec2-user/python-novos/pecas.zip',
as_attachment=True)

        def getCommandByImage(content):
         image = readb64(content)
         r = 1100.0 / image.shape[1]
         dim = (1100, int(image.shape[0] * r))
         image = cv2.resize(image,dim, interpolation = cv2.INTER_AREA)
         pre_proc = Command.preprocess_image(image)
         cnts, qntd_found, qtnd_squard = Command.find_cnts_commands(pre_proc)
         commands = Command.find_commands(cnts, image)
         return Command.responseCommands(commands)

        app.run(host='0.0.0.0', port=80)
```

APÊNDICE B — CODIGO JOGO

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;
using UnityEngine.UI;
using System;

public class BoardManager : MonoBehaviour
{
    public GameObject Collectable, Floor, Obstacle, Player, Wall;

     public GameObject player, circle_title, star_title, triangle_title, circle, star, triangle, loop, left, right, move, _2, _3, _4, _5, _6, _7, _8, _9;

       public Sprite circleMark, starMark, triangleMark, loopMark, leftMark, rightMark, moveMark, _2Mark, _3Mark, _4Mark, _5Mark, _6Mark, _7Mark, _8Mark, _9Mark, loopExecute, playerCrashedSprite;

    private List<Function> functionsBoard;

            private float offsetXBoard, offsetYBoard, offsetXCommand, offsetYCommand, speed = 0.01f;

    private GameManager gameManager;
    private GameObject playerObjectLevel;
    private Rigidbody2D playerBody;
    private Transform playerTransform;

    private Level level;
      private int commands = 0, collectables = 0, indexCircle, indexTriangle, indexStar;

    bool endGame = false;

    PlayerDirection playerDirection;

    Command previous = null;
```

```
        public void initValues()
        {
            offsetXCommand = circle.GetComponent<Image>().sprite.bounds.size.x;
            offsetYCommand = circle.GetComponent<Image>().sprite.bounds.size.y;
                        offsetXBoard = Floor.GetComponent<SpriteRenderer>
().sprite.bounds.size.x;
                        offsetYBoard = Floor.GetComponent<SpriteRenderer>
().sprite.bounds.size.y;
            gameManager = GameManager.instance;
        }

        public void SetupScene(string idLevel)
        {
            commands = 0;
            collectables = 0;
            this.level = JsonUtility.FromJson<Level>(getJsonFileById(idLevel));
            boardSetup(level.board, level.playerDirection);
        }

        public void boardSetup(Board board, string playerDirection)
        {
            Transform boardHolder = GameObject.Find("Board").transform;
            GameObject floor, toInstantiate, instance;
            foreach (DataBoard dataBoard in board.data)
            {
                                                floor = Instantiate(Floor,
getPositionBoardInstance(dataBoard.positionX, dataBoard.positionY),
Quaternion.identity) as GameObject;
                floor.transform.SetParent(boardHolder);
                if (dataBoard.objectType != "Floor")
                {
                    toInstantiate = getObjectToInstantiate(dataBoard.objectType);
                                            instance = Instantiate(toInstantiate,
getPositionBoardInstance(dataBoard.positionX, dataBoard.positionY),
Quaternion.identity) as GameObject;
                    instance.transform.SetParent(boardHolder);
                    if (dataBoard.objectType == "Player")
                    {
```

```
                playerObjectLevel = instance;
                playerBody = instance.GetComponent<Rigidbody2D>();
                playerTransform = instance.transform;
                setPlayerDirection(playerDirection);
            }
        }
    }
}

private void setPlayerDirection(string direction)
{
    direction = direction.ToUpper();
    switch (direction)
    {
        case "UP":
            playerTransform.Rotate(new Vector3(0,0,0));
            playerDirection = PlayerDirection.UP;
            break;
        case "DOWN":
            playerDirection = PlayerDirection.DOWN;
            playerTransform.Rotate(new Vector3(0, 0, -180));
            break;
        case "LEFT":
            playerDirection = PlayerDirection.LEFT;
            playerTransform.Rotate(new Vector3(0, 0, 90));
            break;
        case "RIGHT":
            playerDirection = PlayerDirection.RIGHT;
            playerTransform.Rotate(new Vector3(0, 0, -90));
            break;
    }
}

public StatusGame checkEndGame(int addCommand, int addCollectable)
{
    commands += addCommand;
    collectables += addCollectable;
    if (commands >= level.maxCommands)
```

```
        {
            return StatusGame.DEFEAT;
        }
        else if (collectables >= level.collectable)
        {
            return StatusGame.VICTORY;
        }
        else
        {
            return StatusGame.CONTINUE;
        }
    }

    public int getDifficultHelp()
    {
        return level.difficulty;
    }

    public IEnumerator execute(List<Function> functions)
    {
        functionsBoard = functions;
                                                gameManager.setCommands(functions,
GameObject.Find("BoardCommand").transform, 40, 40, -7f, 0.8f, false);
        yield return StartCoroutine(DoFunction(functions[0]));
        if (!endGame)
        {
            StartCoroutine(gameManager.doDefeat(false));
        }
    }

    private IEnumerator DoFunction(Function function)
    {
        previous = null;
        foreach (Command command in function.Commands)
        {
            animationCommand(command);
            if (!endGame)
            {
                endGame = gameManager.checkEndGameCommand();
```

```
                switch (command.EnumCommand)
                {
                    case EnumCommand.MOVE:
                        yield return StartCoroutine(Move());
                        break;
                    case EnumCommand.CIRCLE:
                        yield return new WaitForSeconds(1f);
                        animationCommand(null);
                                                                yield return
StartCoroutine(DoFunction(functionsBoard[indexCircle]));
                        break;
                    case EnumCommand.STAR:
                        yield return new WaitForSeconds(1f);
                        animationCommand(null);
                                                                yield return
StartCoroutine(DoFunction(functionsBoard[indexStar]));
                        break;
                    case EnumCommand.TRIANGLE:
                        yield return new WaitForSeconds(1f);
                        animationCommand(null);
                                                                yield return
StartCoroutine(DoFunction(functionsBoard[indexTriangle]));
                        break;
                    case EnumCommand.LOOP:
                        yield return StartCoroutine(Loop(command));
                        break;
                    case EnumCommand.LEFT:
                        yield return StartCoroutine(Turn(PlayerDirection.LEFT));
                        break;
                    case EnumCommand.RIGHT:
                        yield return StartCoroutine(Turn(PlayerDirection.RIGHT));
                        break;
                    case EnumCommand.CIRCLE_TITLE:
                        yield return new WaitForSeconds(1f);
                        commands--;
                        break;
                    case EnumCommand.STAR_TITLE:
                        yield return new WaitForSeconds(1f);
```

```
        animationCommand(null);
                        commands--;
                        break;
                    case EnumCommand.TRIANGLE_TITLE:
                        yield return new WaitForSeconds(1f);
                        animationCommand(null);
                        commands--;
                        break;
                }
            }
            else
            {
                StartCoroutine(gameManager.doDefeat(false));
            }
        }
        animationCommand(null);
    }

    private IEnumerator Turn(PlayerDirection direction)
    {
        if (PlayerDirection.LEFT.Equals(direction))
        {
            setPlayerDirectionTurnLeft();
            yield return StartCoroutine(rotationLeft());
        }
        else if (PlayerDirection.RIGHT.Equals(direction))
        {
            setPlayerDirectionTurnRight();
            yield return StartCoroutine(rotationRight());
        }
        yield return new WaitForSeconds(1f);

    }

    private IEnumerator rotationRight()
    {
        var fromAngle = playerTransform.rotation;
            var toAngle = Quaternion.Euler(playerTransform.eulerAngles + new
Vector3(0, 0, -90));
```

```
        for (var t = 0f; t < 1; t += Time.deltaTime / 1)
        {
            playerTransform.rotation = Quaternion.Slerp(fromAngle, toAngle, t);
            yield return null;
        }
    }

    private IEnumerator rotationLeft()
    {
        var fromAngle = playerTransform.rotation;
            var toAngle = Quaternion.Euler(playerTransform.eulerAngles + new
Vector3(0, 0, 90));
        for (var t = 0f; t < 1; t += Time.deltaTime / 1)
        {
            playerTransform.rotation = Quaternion.Slerp(fromAngle, toAngle, t);
            yield return null;
        }
    }

    private void setPlayerDirectionTurnRight()
    {
        if (PlayerDirection.UP.Equals(playerDirection))
        {
            playerDirection = PlayerDirection.RIGHT;
        }
        else if (PlayerDirection.DOWN.Equals(playerDirection))
        {
            playerDirection = PlayerDirection.LEFT;
        }
        else if (PlayerDirection.LEFT.Equals(playerDirection))
        {
            playerDirection = PlayerDirection.UP;
        }
        else if (PlayerDirection.RIGHT.Equals(playerDirection))
        {
            playerDirection = PlayerDirection.DOWN;
        }
    }
```

```
private void setPlayerDirectionTurnLeft()
{
    if (PlayerDirection.UP.Equals(playerDirection))
    {
        playerDirection = PlayerDirection.LEFT;
    }
    else if (PlayerDirection.DOWN.Equals(playerDirection))
    {
        playerDirection = PlayerDirection.RIGHT;
    }
    else if (PlayerDirection.LEFT.Equals(playerDirection))
    {
        playerDirection = PlayerDirection.DOWN;
    }
    else if (PlayerDirection.RIGHT.Equals(playerDirection))
    {
        playerDirection = PlayerDirection.UP;
    }
}

private IEnumerator Loop(Command command)
{
    yield return new WaitForSeconds(1f);
    animationCommand(null);
    Function function = new Function(command.loop);
    animationLoop(command, false);
    for (int x = 0; x < command.numRepeatLoop; x++)
    {
        yield return StartCoroutine(DoFunction(function));
    }
    animationLoop(command, true);
}

private void animationLoop(Command command, bool endAnimation)
{
    if (endAnimation)
    {
```

```
        command.gameObject.GetComponent().sprite = loop.GetComponent().sprite;
                        command.numRepeatLoopGameObject.GetComponent<Image>
().sprite    =    getNumberLoop(command.numRepeatLoop).GetComponent<Image>
().sprite;
            }
            else
            {
                command.gameObject.GetComponent<Image>().sprite = loopExecute;
                        command.numRepeatLoopGameObject.GetComponent<Image>
().sprite = getNumberMarkLoop(command.numRepeatLoop);
            }
            previous = command;
        }

        private IEnumerator Move()
        {
            switch (playerDirection)
            {
                case PlayerDirection.UP:
                    yield return StartCoroutine(MoveUp());
                    break;
                case PlayerDirection.DOWN:
                    yield return StartCoroutine(MoveDown());
                    break;
                case PlayerDirection.LEFT:
                    yield return StartCoroutine(MoveLeft());
                    break;
                case PlayerDirection.RIGHT:
                    yield return StartCoroutine(MoveRight());
                    break;
            }
        }

        private IEnumerator MoveRight()
        {
            Vector2 target = playerBody.position - new Vector2(offsetYBoard, 0);
            target = playerBody.position + new Vector2(offsetYBoard, 0);
            while (playerBody.position.x < target.x)
            {
```

```
                playerBody.MovePosition(new Vector2(playerBody.position.x + speed,
playerBody.position.y));
                yield return null;
            }
            yield return new WaitForSeconds(1);
        }

        private IEnumerator MoveLeft()
        {
            Vector2 target = playerBody.position - new Vector2(offsetYBoard, 0);
            while (playerBody.position.x > target.x)
            {
                playerBody.MovePosition(new Vector2(playerBody.position.x - speed,
playerBody.position.y));
                yield return null;
            }
            yield return new WaitForSeconds(1);
        }

        private IEnumerator MoveDown()
        {
            Vector2 target = playerBody.position - new Vector2(0, offsetXBoard);
            while (playerBody.position.y > target.y)
            {
                        playerBody.MovePosition(new  Vector2(playerBody.position.x,
playerBody.position.y - speed));
                yield return null;
            }
            yield return new WaitForSeconds(1);

        }

        private IEnumerator MoveUp()
        {
            Vector2 target = playerBody.position + new Vector2(0, offsetYBoard);
            while (playerBody.position.y < target.y)
            {
                        playerBody.MovePosition(new  Vector2(playerBody.position.x,
```

```
playerBody.position.y + speed));
                yield return null;
            }
            yield return new WaitForSeconds(1);
        }

        private void animationCommand(Command command)
        {
            if (previous != null)
            {
                                                    GameObject instance =
getObjectToInstantiate(previous.EnumCommand);
                                previous.gameObject.GetComponent<Image>().sprite =
instance.GetComponent<Image>().sprite;
            }
            if (command != null)
            {
                                command.gameObject.GetComponent<Image>().sprite =
getSpriteCommandMark(command.EnumCommand);
            }
            previous = command;
        }

        public IEnumerator terminateMovement(Vector2 positionCollectable)
        {
            while (!playerBody.position.Equals(positionCollectable))
            {
                    playerBody.position = Vector3.MoveTowards(playerBody.position,
positionCollectable, speed);
                yield return null;
            }
        }

        private GameObject getObjectToInstantiate(string objectType)
        {
            switch (objectType)
            {
                case "Collectable":
                    level.collectable++;
```

```
            return Collectable;
        case "Floor":
            return Floor;
        case "Obstacle":
            return Obstacle;
        case "Player":
            return Player;
        case "Wall":
            return Wall;
        default:
            return Floor;
    }
}

private Vector3 getPositionBoardInstance(int x, int y)
{
    float temp = y + 0.8f;
    return new Vector3(x * offsetXBoard, temp * offsetYBoard, 0f);
}

public Vector3 getPositionCommandInstance(int x, float y)
{
    return new Vector3(x * offsetXCommand, y * offsetYCommand, 0f);
}

private Sprite getSpriteCommandMark(EnumCommand enumCommand)
{
    switch (enumCommand)
    {


        case EnumCommand.CIRCLE:
            return circleMark;
        case EnumCommand.CIRCLE_TITLE:
            return circleMark;
        case EnumCommand.STAR:
            return starMark;
        case EnumCommand.STAR_TITLE:
```

```
return starMark;
            case EnumCommand.TRIANGLE:
                return triangleMark;
            case EnumCommand.TRIANGLE_TITLE:
                return triangleMark;
            case EnumCommand.LOOP:
                return loopMark;
            case EnumCommand.LEFT:
                return leftMark;
            case EnumCommand.RIGHT:
                return rightMark;
            case EnumCommand.MOVE:
                return moveMark;
            default:
                return null;
        }
    }

    public Sprite getNumberMarkLoop(int number)
    {
        switch (number)
        {
            case 2:
                return _2Mark;
            case 3:
                return _3Mark;
            case 4:
                return _4Mark;
            case 5:
                return _5Mark;
            case 6:
                return _6Mark;
            case 7:
                return _7Mark;
            case 8:
                return _8Mark;
            case 9:
                return _9Mark;
            default:
```

```
            return null;
        }
    }


    private string getJsonFileById(string idLevel)
    {
        TextAsset file = Resources.Load("Level-" + idLevel) as TextAsset;
        string json = file.ToString();
        return json;
    }

    public string getCommandsRemaining()
    {
                                                                    return
string.Format(GameManager.instance.messages.getLabelMovimentos(), commands,
level.maxCommands);
    }

    public void setIndex(int indexCircle, int indexStar, int indexTriangle)
    {
        this.indexCircle = indexCircle;
        this.indexStar = indexStar;
        this.indexTriangle = indexTriangle;
    }

    public GameObject getNumberLoop(int number)
    {
        switch (number)
        {
            case 2:
                return _2;
            case 3:
                return _3;
            case 4:
                return _4;
            case 5:
                return _5;
```

```
case 6:
            return _6;
        case 7:
            return _7;
        case 8:
            return _8;
        case 9:
            return _9;
        default:
            return null;
    }
}

public GameObject getObjectToInstantiate(EnumCommand command)
{
    switch (command)
    {
        case EnumCommand.CIRCLE:
            return circle;
        case EnumCommand.CIRCLE_TITLE:
            return circle_title;
        case EnumCommand.STAR:
            return star;
        case EnumCommand.STAR_TITLE:
            return star_title;
        case EnumCommand.TRIANGLE:
            return triangle;
        case EnumCommand.TRIANGLE_TITLE:
            return triangle_title;
        case EnumCommand.LOOP:
            return loop;
        case EnumCommand.LEFT:
            return left;
        case EnumCommand.RIGHT:
            return right;
        case EnumCommand.MOVE:
            return move;
        default:
            return null;
```

```
            }
        }

        public int getMaxPiece()
        {
            return level.maxCommandsUse;
        }

        public void playerCrashed()
        {
                    playerObjectLevel.GetComponent<SpriteRenderer>().sprite =
playerCrashedSprite;
        }
    }

    using System.Collections.Generic;
    using UnityEngine;

    public enum EnumCommand
    {
        CIRCLE,
        CIRCLE_TITLE,
        STAR,
        STAR_TITLE,
        TRIANGLE,
        TRIANGLE_TITLE,
        LOOP,
        LEFT,
        RIGHT,
        MOVE,
        UNKNOW
    };

    public class Languages
    {
        public static string languagePTBR = "pt-BR";
        public static string languageENUS = "en-US";
    }
```

```
public enum PlayerDirection
{
    UP,
    DOWN,
    LEFT,
    RIGHT
}

public enum StatusGame
{
    VICTORY,
    DEFEAT,
    CONTINUE
}

public class Function
{
    public List<Command> Commands;

    public Function(List<Command> commands)
    {
        this.Commands = commands;
    }
}

public class Command
{
    public EnumCommand EnumCommand;
    public GameObject gameObject;
    public List<Command> loop;
    public int numRepeatLoop;
    public GameObject numRepeatLoopGameObject;


    public Command(EnumCommand command)
    {
        this.EnumCommand = command;
    }
```

```
}

[System.Serializable]
public class Level
{
    public string name;
    public string levelID;
    public string author;
    public int difficulty;
    public int collectable;
    public int maxCommands;
    public int maxCommandsUse;
    public string playerDirection;
    public Board board;
}

[System.Serializable]
public class Board
{
    public DataBoard[] data;
}

[System.Serializable]
public class DataBoard
{
    public int positionX;
    public int positionY;
    public string objectType;
}

using UnityEngine;
using System.Collections;
using UnityEngine.SceneManagement;
using System;
using System.Collections.Generic;
using UnityEngine.UI;
```

```
public class GameManager : MonoBehaviour
{

    public static GameManager instance = null;

    private BoardManager boardScript;
    private LevelManager levelManager;
    private SelectLevelManager selectLevelManager;
    private String levelId;
    public int maxLevel;
    public Messages messages;

    public ModalPanelManager ModalPanelManager;
    public ModalPanelHelpManager ModalPanelHelpManager;

    private List<Function> functions;

    void Awake()
    {
        if (instance == null)
            instance = this;
        else if (instance != this)
            Destroy(gameObject);
        DontDestroyOnLoad(gameObject);
        boardScript = GetComponent<BoardManager>();
        boardScript.initValues();
        messages = getLanguage();
    }

    void Start()
    {
        int firstAcess = PlayerPrefs.GetInt("firstAcess", 1);
        if (firstAcess == 1)
        {
            clickHelp(-1);
        }
        PlayerPrefs.SetInt("firstAcess", 0);
    }
```

```
        private Messages getLanguage()
        {
                        string language = PlayerPrefs.GetString("language",
Languages.languagePTBR);
            if (language.Equals(Languages.languagePTBR))
            {
                return new PT_BR();
            }
            else
            {
                return new EN_US();
            }
        }

        public void soundClick()
        {
            SoundManager.instance.soundClick();
        }

        public void pictureClick()
        {
            RecognizeCommandManager.instance.pictureClick();
        }

        public void takePictureClick()
        {

RecognizeCommandManager.instance.takePictureClick(GameManager.instance.boa
rdScript.getMaxPiece());
        }

        public void recognizeCommand(List<Function> functions, int indexCircle, int
indexStar, int indexTriangle)
        {
            this.functions = functions;
            ModalPanelManager.setCommands(functions, this);
            ModalPanelManager.activeModal("", false, false, true, false, false);

ModalPanelManager.setTitleCommands(messages.getTituloPainelComandos());
```

```
        boardScript.setIndex(indexCircle, indexStar, indexTriangle);
        }

                public void showErro(string erro, bool buttonOkVisible, bool
buttonTryAgainVisible)
        {
            ModalPanelManager.activeModal(messages.getTituloPainelErro(), false,
true, false, false, false);
            ModalPanelManager.setDescriptionError(erro);
                        ModalPanelManager.setVisibleButtonsErro(buttonOkVisible,
buttonTryAgainVisible);
        }

        public void functionsCorrect()
        {
            GameManager gameManager = GameManager.instance;
            gameManager.ModalPanelManager.deactiveModal();

gameManager.StartCoroutine(gameManager.boardScript.execute(gameManager.fun
ctions));
            gameManager.levelManager.deactivateButtons();
        }

        public void functionsWrong()
        {
            GameManager gameManager = GameManager.instance;
            gameManager.ModalPanelManager.deactiveModal();
            gameManager.pictureClick();
        }

        public void loadLevel(String id)
        {
            GameManager gameManager = GameManager.instance;
            if (id.Equals("-1")) {
                int newLevel = Int32.Parse(gameManager.levelId) + 1;
                gameManager.levelId = newLevel.ToString();
            }
            else
            {
                gameManager.levelId = id;
```

```
            }
            gameManager.loadScene(2);
        }

        public void setupSceneLevel(LevelManager levelManager)
        {
            this.levelManager = levelManager;
            this.boardScript.SetupScene(levelId);

levelManager.setTextCommands(boardScript.getCommandsRemaining());

levelManager.setMaxPieces(string.Format(messages.getMaxUse(),boardScript.getM
axPiece()));
            levelManager.setTitle(levelId);
            levelManager.setHelp(boardScript.getDifficultHelp());
        }

        public void loadScene(int sceneIndex)
        {
            SceneManager.LoadScene(sceneIndex);
        }

        public Boolean checkEndGameCommand()
        {
            StatusGame status = boardScript.checkEndGame(1, 0);

levelManager.setTextCommands(boardScript.getCommandsRemaining());
            if (status.Equals(StatusGame.DEFEAT))
            {
                StartCoroutine(doDefeat(false));
                return true;
            }
            return false;
        }

        public void clickHelp(int helpDifficult)
        {
            GameManager gameManager = GameManager.instance;
            gameManager.ModalPanelHelpManager.activeModal(helpDifficult);
```

```
    }

    public void clickCloseModal()
    {
        GameManager.instance.ModalPanelManager.deactiveModal();
    }

    public void clickCloseModalHelp()
    {
        GameManager.instance.ModalPanelHelpManager.deactiveModal();
    }

    public void reloadLevel()
    {
        GameManager.instance.boardScript.StopAllCoroutines();
        SceneManager.LoadScene(2);
    }

    public void checkEndGameCollectable(Vector2 positionCollectable)
    {
        StatusGame status = boardScript.checkEndGame(0, 1);
        if (status.Equals(StatusGame.VICTORY))
        {
            StartCoroutine(doVictory(positionCollectable));
        }
    }

    public IEnumerator doDefeat(Boolean isCrashed)
    {
        boardScript.StopAllCoroutines();
        if (isCrashed)
        {
            boardScript.playerCrashed();
            yield return new WaitForSeconds(0.5f);
        }

ModalPanelManager.activeModal(messages.getTituloPainelFimJogoDerrota(),    true,
false, false, false, false);
            ModalPanelManager.interactableButtonNext(false);
```

```
        }

        private IEnumerator doVictory(Vector2 positionCollectable)
        {
            boardScript.StopAllCoroutines();
                                                                yield return
StartCoroutine(boardScript.terminateMovement(positionCollectable));

            int level = Int32.Parse(levelId);

            if (level < maxLevel)
            {

ModalPanelManager.activeModal(messages.getTituloPainelFimJogoVitoria(), true,
false, false, false, false);
            }
            else
            {

ModalPanelManager.activeModal(messages.getTituloPainelFimJogoVitoria(), false,
false, false, true, false);

ModalPanelManager.setDescriptionLastLevel(messages.getMensagemUltimaFase())
;
            }
            int levelReached = PlayerPrefs.GetInt("levelReached", 1);
            if (level >= levelReached)
            {
                PlayerPrefs.SetInt("levelReached", level+1);
            }

            ModalPanelManager.interactableButtonNext(true);
        }

                public void setCommands(List<Function> functions, Transform
transformBoardCommand, int width, int height, float diffX, float diffY, bool
clearTransform)
        {
            if (clearTransform)
            {
```

```
            foreach (Transform child in transformBoardCommand)
            {
                GameObject.Destroy(child.gameObject);
            }
        }

        for (int y = 0; y < functions.Count; y++)
        {
            int positionBoard = 0;
            for (int x = 0; x < functions[y].Commands.Count; x++)
            {
                                printCommandOnBoard(functions[y].Commands[x], ref
positionBoard, y, transformBoardCommand.transform, false, width, height, diffX,
diffY);
                                                                        if
(EnumCommand.LOOP.Equals(functions[y].Commands[x].EnumCommand))
                {
                    for (int a = 0; a < functions[y].Commands[x].loop.Count; a++)
                    {
                        printCommandOnBoard(functions[y].Commands[x].loop[a], ref
positionBoard, y, transformBoardCommand.transform, false, width, height, diffX,
diffY);
                    }
                                printCommandOnBoard(functions[y].Commands[x], ref
positionBoard, y, transformBoardCommand.transform, true, width, height, diffX, diffY);
                }

            }
        }
    }

     private void printCommandOnBoard(Command command, ref int positionX,
int positionY, Transform transform, bool numberRepeat, int width, int height, float
diffX, float diffY)
    {
        GameObject toInstantiate, commandObject;
        if (numberRepeat)
        {
```

```
        toInstantiate = boardScript.getNumberLoop(command.numRepeatLoop);
            }
            else
            {
                                                                toInstantiate =
boardScript.getObjectToInstantiate(command.EnumCommand);
            }
            commandObject = new GameObject();
            Image image = commandObject.AddComponent<Image>();
                                    commandObject.GetComponent<RectTransform>
().SetParent(transform.transform, false);
                commandObject.GetComponent<RectTransform>().sizeDelta = new
Vector2(width, height);
            image.sprite = toInstantiate.GetComponent<Image>().sprite;
             commandObject.transform.localPosition = getPositionInstance(positionX
+ diffX, (positionY * -1) + diffY, width, height);
            if (!numberRepeat) {
               command.gameObject = commandObject;
            }
            else
            {
               command.numRepeatLoopGameObject = commandObject;
            }
            positionX++;
        }

            private Vector3 getPositionInstance(float x, float y, float offsetX, float
offsetY)
        {
            return new Vector3(x * offsetX - 5, y * offsetY - 5, 0f);
        }

        public void languageClick(int scene)
        {
            GameManager gameManager = GameManager.instance;

gameManager.ModalPanelManager.activeModal(gameManager.messages.getTituloP
ainelEscolherLinguagem(), false, false, false, false, true);
            gameManager.ModalPanelManager.Scene = scene;
```

```
        }

        public void languageSelect(int languageSelect)
        {
            GameManager gameManager = GameManager.instance;
                                string language = PlayerPrefs.GetString("language",
Languages.languagePTBR);
             if (!language.Equals(Languages.languagePTBR) && languageSelect ==
0)
            {
                gameManager.messages = new PT_BR();
                gameManager.loadScene(gameManager.ModalPanelManager.Scene);
                PlayerPrefs.SetString("language", Languages.languagePTBR);
            }
             else if(!language.Equals(Languages.languageENUS) && languageSelect
== 1)
            {
                gameManager.messages = new EN_US();
                gameManager.loadScene(gameManager.ModalPanelManager.Scene);
                PlayerPrefs.SetString("language", Languages.languageENUS);
            }
            gameManager.ModalPanelManager.deactiveModal();
        }

        public void setSelectLevel(SelectLevelManager selectLevelManager)
        {
            this.selectLevelManager = selectLevelManager;
        }

        public void nextPageLevelClick()
        {
            GameManager.instance.selectLevelManager.nextPageLevelClick();
        }

        public void backPageSelectLevelClick()
        {

GameManager.instance.selectLevelManager.backPageSelectLevelClick();
```

```
    }

        public void downloadPieces()
        {
                                Application.OpenURL("http://ec2-18-224-2-172.us-east-
2.compute.amazonaws.com/pecas");
        }
    }

    using System;
    using System.Collections.Generic;
    using UnityEngine;
    using UnityEngine.UI;

    public class RecognizeCommandManager : MonoBehaviour
    {
        public static RecognizeCommandManager instance = null;
        private CameraViewModalManager cameraViewManager;
        private int maxCommandsUse;
        PhoneCamera phoneCamera;

        void Awake()
        {
            if (instance == null)
                instance = this;
            else if (instance != this)
                Destroy(gameObject);

            DontDestroyOnLoad(gameObject);
        }

        public void pictureClick()
        {
                                                    phoneCamera = new
PhoneCamera(cameraViewManager.GetComponent<RawImage>());
            cameraViewManager.active();
        }

        public void takePictureClick(int maxCommandsUse)
```

```
        {
            cameraViewManager.loading();
            byte[] bytes = phoneCamera.TakePhoto();
            this.maxCommandsUse = maxCommandsUse;

//response("star,loop,left,move,move,3,circle,next,circle,loop,move,2,NEXT,triangle,ri
ght", false);
            StartCoroutine(RequestManager.Request(bytes, this));
        }

                public void setCameraViewManager(CameraViewModalManager
cameraViewManager)
        {
            this.cameraViewManager = cameraViewManager;
        }


        public void response(string response, bool error)
        {
            cameraViewManager.deactivate();
            if (!error)
            {
                convertToCommand(response);
            }
            else
            {
                GameManager.instance.showErro(response, false, true);
            }
        }

        public void convertToCommand(string response)
        {
            GameManager gameManager = GameManager.instance;
             bool error = false, firstCommandInLine = true, refCircle = false, refStar =
false, refTriangle = false, loop = false;
            if (response.ToUpper().Equals("UNKNOWN"))
            {
```

```
        gameManager.showErro(gameManager.messages.getErroNenhumComando
Reconhecido(), false, true);
                error = true;
            }
            response = response.ToUpper();
            string[] commands = response.Split(',');
            List<Function> functions= new List<Function>();
                int line = 0, indexCircle = -1, indexStar = -1, indexTriangle = -1,
commandCount = 0;
            Command commandLoop = new Command(EnumCommand.LOOP);
                    List<Command> commandsLine = new List<Command>(),
commandsLoop = new List<Command>();
            Command command;

            foreach (string commandString in commands)
            {
               commandCount++;
               if (!error)
               {
                  command = getCommand(commandString);
                  if (firstCommandInLine)
                  {
                     error = firstCommandLineCheckError(command.EnumCommand,
ref indexCircle, ref indexStar, ref indexTriangle, line);
                     firstCommandInLine = false;
                     commandsLine.Add(title(command.EnumCommand));
                  }
                  else if (commandString.Equals("NEXT"))
                  {
                                          error = nextLineCheckError(line, loop,
getFuncionInLine(line,indexCircle,indexTriangle,indexStar));
                     line++;
                     functions.Add(new Function(commandsLine));
                     commandsLine = new List<Command>();
                     firstCommandInLine = true;
                     commandCount--;
                  }
                  else if (command.EnumCommand.Equals(EnumCommand.LOOP))
                  {
```

```
                if (loop)
                {

gameManager.showErro(String.Format(gameManager.messages.getErroLoop(),
getFuncionInLine(line, indexCircle, indexTriangle, indexStar)), false, true);
                    error = true;
                }
                else
                {
                    loop = true;
                    commandLoop = command;
                    commandsLoop = new List<Command>();
                }
            }
                                                                    else if
(command.EnumCommand.Equals(EnumCommand.UNKNOW)                     &&
commandNumber(commandString))
            {
                if (loop)
                {
                    if (commandsLoop.Count > 0) {
                        commandLoop.loop = commandsLoop;
                                                commandLoop.numRepeatLoop =
numRepeat(commandString);
                        loop = false;
                        commandsLine.Add(commandLoop);
                    }
                    else
                    {

gameManager.showErro(String.Format(gameManager.messages.getErroLoopSemC
omando(), getFuncionInLine(line, indexCircle, indexTriangle, indexStar)), false, true);
                        error = true;
                    }
                }
                else
                {
```

```
        gameManager.showErro(String.Format(gameManager.messages.getErroNum
ero(), getFuncionInLine(line, indexCircle, indexTriangle, indexStar)), false, true);
                        error = true;
                    }
                }
                                                                    else      if
(command.EnumCommand.Equals(EnumCommand.UNKNOW))
                {

gameManager.showErro(gameManager.messages.getErroAoReconhecerComando()
, false, true);
                    error = true;
                }
                else
                {
                    if (loop)
                    {
                        commandsLoop.Add(command);
                    }
                    else {
                        commandsLine.Add(command);
                    }
                }
                  refFuncion(command.EnumCommand, ref refCircle, ref refTriangle,
ref refStar);
            }
        }
        if (!error)
        {
            functions.Add(new Function(commandsLine));
                            if  (!checkErrorCommandUse(commandCount,  line)  &&
!checkErrorFucntionReference(indexCircle,  indexTriangle,  indexStar,  refCircle,
refStar, refTriangle) &&
                !checkLoop(loop, getFuncionInLine(line, indexCircle, indexTriangle,
indexStar)))
            {
                        gameManager.recognizeCommand(functions,  indexCircle,
indexStar, indexTriangle);
            }
```

```
        }
    }

    private string getFuncionInLine(int line, int circle, int triangle, int star)
    {
        Messages messages = GameManager.instance.messages;
        if(line == circle)
        {
            return messages.getFuncaoCirculo();
        }
        else if(line == triangle)
        {
            return messages.getFuncaoTriangulo();
        }
        else
        {
            return messages.getFuncaoEstrela();
        }
    }

    private int numRepeat(string commandString)
    {
        int r = -1;
        int.TryParse(commandString, out r);
        return r;
    }

    private bool commandNumber(string commandString)
    {
        int num = numRepeat(commandString);
        return (num > 1 && num <= 9);
    }

    private void refFuncion(EnumCommand enumCommand, ref bool refCircle,
ref bool refTriangle, ref bool refStar)
    {
        if (!refCircle && enumCommand.Equals(EnumCommand.CIRCLE))
        {
```

```
        refCircle = true;
            }
                                                    else if (!refTriangle &&
enumCommand.Equals(EnumCommand.TRIANGLE))
            {
                refTriangle = true;
            }
            else if (!refStar && enumCommand.Equals(EnumCommand.STAR))
            {
                refStar = true;
            }
        }

        private bool nextLineCheckError(int line, bool loop, string function)
        {
            GameManager gameManager = GameManager.instance;
            if (line == 2)
            {

GameManager.instance.showErro(String.Format(gameManager.messages.getErroLi
nhasInvalidas(), line+1), false, true);
                return true;
            }
            return checkLoop(loop, function);
        }

        private bool checkLoop(bool loop, string function)
        {
            if (loop)
            {

GameManager.instance.showErro(String.Format(GameManager.instance.messages.
getErroLoop(), function), false, true);
                return true;
            }
            return false;
        }
         private bool checkErrorFucntionReference(int indexCircle, int indexTriangle,
int indexStar, bool refCircle, bool refStar, bool refTriangle)
```

```
        {
            GameManager gameManager = GameManager.instance;
            if (refCircle)
            {
                if(indexCircle == -1)
                {

GameManager.instance.showErro(String.Format(gameManager.messages.getErroFu
ncaoNaoImplementada(), gameManager.messages.getFuncaoCirculo()), false, true);
                    return true;
                }
            }
            if (refStar)
            {
                if (indexStar == -1)
                {

GameManager.instance.showErro(String.Format(gameManager.messages.getErroFu
ncaoNaoImplementada(), gameManager.messages.getFuncaoEstrela()), false, true);
                    return true;
                }
            }
            if (refTriangle)
            {
                if (indexTriangle == -1)
                {

GameManager.instance.showErro(String.Format(gameManager.messages.getErroFu
ncaoNaoImplementada(), gameManager.messages.getFuncaoTriangulo()), false,
true);
                    return true;
                }
            }
            return false;
        }

        private bool checkErrorCommandUse(int commands, int line)
        {
            if (maxCommandsUse < commands)
```

```
        {

GameManager.instance.showErro(String.Format(GameManager.instance.messages.
getErroQuantidadeComandos(), maxCommandsUse, commands), false, true);
                return true;
            }
            return false;
        }

        private Command title(EnumCommand enumCommand)
        {
            if (enumCommand.Equals(EnumCommand.CIRCLE))
            {
                return new Command(EnumCommand.CIRCLE_TITLE);
            }
            else if (enumCommand.Equals(EnumCommand.TRIANGLE))
            {
                return new Command(EnumCommand.TRIANGLE_TITLE);
            }
            else
            {
                return new Command(EnumCommand.STAR_TITLE);
            }
        }

                    private bool firstCommandLineCheckError(EnumCommand
enumCommand, ref int indexCircle, ref int indexStar, ref int indexTriangle, int line)
        {
            GameManager gameManager = GameManager.instance;
            if (EnumCommand.CIRCLE.Equals(enumCommand))
            {
                if(indexCircle > -1)
                {

gameManager.showErro(String.Format(gameManager.messages.getErroFuncaoDefi
nidaDuasVezes(), gameManager.messages.getFuncaoCirculo()), false, true);
                    return true;
                }
                else
```

```
            {
                indexCircle = line;
            }
        }else if (EnumCommand.TRIANGLE.Equals(enumCommand))
        {
            if (indexTriangle > -1)
            {

gameManager.showErro(String.Format(gameManager.messages.getErroFuncaoDefi
nidaDuasVezes(), gameManager.messages.getFuncaoTriangulo()), false, true);
                return true;
            }
            else
            {
                indexTriangle = line;
            }
        }
        else if (EnumCommand.STAR.Equals(enumCommand))
        {
            if (indexStar > -1)
            {

gameManager.showErro(String.Format(gameManager.messages.getErroFuncaoDefi
nidaDuasVezes(), gameManager.messages.getFuncaoEstrela()), false, true);
                return true;
            }
            else
            {
                indexStar = line;
            }
        }
        else
        {

gameManager.showErro(String.Format(gameManager.messages.getPrimeiroComan
doLinha(), line+1), false, true);
            return true;
        }
```

```
        return false;
        }

        public Command getCommand(string command)
        {
            switch (command)
            {
                case "CIRCLE":
                    return new Command(EnumCommand.CIRCLE);
                case "STAR":
                    return new Command(EnumCommand.STAR);
                case "TRIANGLE":
                    return new Command(EnumCommand.TRIANGLE);
                case "LOOP":
                    return new Command(EnumCommand.LOOP);
                case "LEFT":
                    return new Command(EnumCommand.LEFT);
                case "RIGHT":
                    return new Command(EnumCommand.RIGHT);
                case "MOVE":
                    return new Command(EnumCommand.MOVE);
                default:
                    return new Command(EnumCommand.UNKNOW);
            }
        }
    }

    using System.Collections;
    using System.Text;
    using UnityEngine.Networking;

    public class RequestManager{

                        public static IEnumerator Request(byte[] bytes,
RecognizeCommandManager recognizeCommandManager)
        {
            string json = getJsonRequest(getImage64(bytes));
              UnityWebRequest www = UnityWebRequest.Post("http://ec2-18-224-2-
172.us-east-2.compute.amazonaws.com", json);
```

```
            byte[] bytesRequest = Encoding.UTF8.GetBytes(json);
            UploadHandlerRaw uH = new UploadHandlerRaw(bytesRequest);
            uH.contentType = "application/json";
            www.uploadHandler = uH;
            {
                yield return null;
                yield return www.SendWebRequest();
                if (www.isNetworkError)
                {

recognizeCommandManager.response(GameManager.instance.messages.getErroPr
oblemaConexao(), true);
                }
                else if(www.isHttpError || www.responseCode != 200){

recognizeCommandManager.response(string.Format(GameManager.instance.messa
ges.getErroServidor(), www.responseCode, www.error), true);
                }
                else
                {
                    recognizeCommandManager.response(www.downloadHandler.text,
false);
                }
            }
        }

        private static string getImage64(byte[] data)
        {
            return System.Convert.ToBase64String(data);
        }

        private static string getJsonRequest(string imageBase64)
        {
            return "{\"image\": \"" + imageBase64 + "\"}";
        }
    }

    using UnityEngine;
```

```
using UnityEngine.UI;

public class CameraViewModalManager : UnityEngine.MonoBehaviour
{
    public Button takePicture;
    public Text loadingText;

    void Start () {
        deactivate();
        RecognizeCommandManager.instance.setCameraViewManager(this);
                                                    loadingText.text =
GameManager.instance.messages.getLabelCarregando();
    }

    public void active()
    {
        gameObject.SetActive(true);
        loadingText.gameObject.SetActive(false);
        takePicture.interactable = true;
    }

    public void deactivate()
    {
        gameObject.SetActive(false);
    }

    public void loading()
    {
        loadingText.gameObject.SetActive(true);
        takePicture.interactable = false;
    }
}

using UnityEngine;

public class Collectable : MonoBehaviour
{
 void Update () {
        transform.Rotate(new Vector3(0, 0, 50) * Time.deltaTime);
```

```
    }
}

using System;
using UnityEngine;
using UnityEngine.UI;

public class LevelManager: MonoBehaviour
{
    public Button picture, help, sound, language, download;
                public Text TextCommands, TextLoading, TextTitle,
TextTitleBoardCommand, TextMaxPieces;

    void Start()
    {
        GameManager gameManager = GameManager.instance;
        gameManager.setupSceneLevel(this);
        TextLoading.text = gameManager.messages.getLabelCarregando();
                                                TextTitleBoardCommand.text =
gameManager.messages.getTituloBoardComandos();
    }

    public void deactivateButtons()
    {
        picture.interactable = false;
        help.interactable = false;
        sound.interactable = false;
        language.interactable = false;
        download.interactable = false;
    }

    public void setTextCommands(string commands)
    {
        TextCommands.text = commands;
    }

    public void setTitle(string level)
    {
```

```
        TextTitle.text = string.Format(GameManager.instance.messages.getTituloTelaFases(), level);
        }

        public void setMaxPieces(string text)
        {
            TextMaxPieces.text = text;
        }

        public void setHelp(int helpDifficult)
        {
                help.onClick.AddListener(delegate { GameManager.instance.clickHelp(helpDifficult); });
        }
    }

    using UnityEngine;

    public class MainPanelScript : MonoBehaviour {


        void Start () {

        }
    }

    using UnityEngine;
    using UnityEngine.UI;

    public class ModalPanelHelpManager : MonoBehaviour {

            public Text title, descriptionHelp, descriptionMove1, descriptionMove2, descriptionMove3, descriptionLoop, descriptionRecursion, descriptionFunction, descriptionButtonDownload, descriptionHelpDownload;
        private GameManager gameManager;
                public GameObject panelHelp, panelHelpMove, panelHelpLoop, panelHelpRecursion, panelFunction, panelDownload;
```

```
    void Start () {
        gameManager = GameManager.instance;
        gameManager.ModalPanelHelpManager = this;
        deactiveModal();
        title.text = gameManager.messages.getTituloPainelAjuda();
      }

      public void deactiveModal()
      {
        gameObject.SetActive(false);
      }

      public void activeModal(int helpDifficult)
      {
        gameObject.SetActive(true);
        panelHelp.SetActive(false);
        panelHelpMove.SetActive(false);
        panelHelpLoop.SetActive(false);
        panelHelpRecursion.SetActive(false);
        panelFunction.SetActive(false);
        panelDownload.SetActive(false);
        switch (helpDifficult)
        {
          case -1:
            panelDownload.SetActive(true);
                                                descriptionButtonDownload.text =
gameManager.messages.getBotaoBaixar();
                                                  descriptionHelpDownload.text =
gameManager.messages.getDescricaoAjudaBaixar();
            break;
          case 0:
                                                          descriptionHelp.text =
gameManager.messages.getDescricaoAjudaSobreJogo();
            panelHelp.SetActive(true);
            break;
          case 1:
                                                        descriptionMove1.text =
gameManager.messages.getDescricaoAjudaMovimentosBasicos1();
```

```
        descriptionMove2.text                                                    =
gameManager.messages.getDescricaoAjudaMovimentosBasicos2();
                                                                descriptionMove3.text    =
gameManager.messages.getDescricaoAjudaMovimentosBasicos3();
                    panelHelpMove.SetActive(true);
                    break;
                case 2:
                    panelHelp.SetActive(true);
                                                                  descriptionHelp.text    =
gameManager.messages.getDescricaoAjudaMaisColetaveis();
                    break;
                case 3:
                    panelFunction.SetActive(true);
                                                              descriptionFunction.text    =
gameManager.messages.getDescricaoAjudaMaisFuncoes();
                    break;
                case 4:
                    panelHelpLoop.SetActive(true);
                                                                  descriptionLoop.text    =
gameManager.messages.getDescricaoAjudaRepeticao();
                    break;
                case 5:
                    panelHelpRecursion.SetActive(true);
                                                             descriptionRecursion.text    =
gameManager.messages.getDescricaoAjudaRecursao();
                    break;
            }
        }

    }

    using System;
    using System.Collections.Generic;
    using UnityEngine;
    using UnityEngine.UI;

    public class ModalPanelManager : MonoBehaviour {
```

```
    public Text title, descriptionError, titleCommands, descriptionLastLevel,
descriptionButtonFases, descriptionButtonTryAgain, descriptionButtonNext;
                public Text descriptionTryAgainError, descriptionOkError,
descriptionButtonYes, descriptionButtonNoTryAgain, descriptionLevelPanelLastLevel,
languageBR, languageUS;
                public GameObject panelEndGame, panelErrorCommand,
panelCommands, panelLastLevel, panelLanguage, icLock;

        public GameObject boardCommand;
        public Button buttonNext, buttonTryAgainError, buttonOkError;

        public int Scene;

        void Start()
        {
            GameManager gameManager = GameManager.instance;
            gameManager.ModalPanelManager = this;
            deactiveModal();
                                                    descriptionButtonFases.text      =
gameManager.messages.getBotaoFases();
                                                descriptionButtonTryAgain.text      =
gameManager.messages.getBotaoTentarNovamente();
                                                    descriptionButtonNext.text      =
gameManager.messages.getBotaoProximaFase();
                                                    descriptionTryAgainError.text      =
gameManager.messages.getBotaoTentarNovamente();
            descriptionOkError.text = gameManager.messages.getBotaoOk();
            descriptionButtonYes.text = gameManager.messages.getBotaoSim();
                                                descriptionButtonNoTryAgain.text      =
gameManager.messages.getNaoBotaoTentarNovamente();
                                            descriptionLevelPanelLastLevel.text      =
gameManager.messages.getBotaoFases();
            languageBR.text = gameManager.messages.getPortugues();
            languageUS.text = gameManager.messages.getIngles();
        }

        public void interactableButtonNext(Boolean interactable)
        {
            buttonNext.interactable = interactable;
```

```
            icLock.SetActive(!interactable);
        }

            public void setCommands(List<Function> functions, GameManager
gameManager)
        {
            gameManager.setCommands(functions, boardCommand.transform, 44,
44, -5, 1, true);
        }

                public void setVisibleButtonsErro(bool buttonOkVisible, bool
buttonTryAgainVisible)
        {
            buttonTryAgainError.gameObject.SetActive(buttonTryAgainVisible);
            buttonOkError.gameObject.SetActive(buttonOkVisible);



        }

            public void activeModal(string title, Boolean panelEndGame, Boolean
panelErrorCommand, Boolean panelCommands, Boolean panelLastLevel, Boolean
panelLanguage)
        {
            gameObject.SetActive(true);
            this.title.text = title;
            this.panelEndGame.SetActive(panelEndGame);
            this.panelErrorCommand.SetActive(panelErrorCommand);
            this.panelCommands.SetActive(panelCommands);
            this.panelLastLevel.SetActive(panelLastLevel);
            this.panelLanguage.SetActive(panelLanguage);
        }

        public void setTitleCommands(string titleCommands)
        {
            this.titleCommands.text = titleCommands;
        }

        public void setDescriptionError(string description)
```

```
{
        descriptionError.text = description;
    }

    public void setDescriptionLastLevel(string description)
    {
        descriptionLastLevel.text = description;
    }

    public void deactiveModal()
    {
        gameObject.SetActive(false);
    }

}

using UnityEngine;
using UnityEngine.UI;

public class PhoneCamera : MonoBehaviour {

    private WebCamTexture cam;

  public PhoneCamera(RawImage cameraImage) {
        WebCamDevice[] devices = WebCamTexture.devices;
        GameManager gameManager = GameManager.instance;
        if(devices.Length == 0)
        {

gameManager.showErro(gameManager.messages.getErroNenhumaCameraEncontrada(), false, true);
        }

        for(int i = 0; i < devices.Length; i++)
        {
            if (!devices[i].isFrontFacing)
            {
                    cam = new WebCamTexture(devices[i].name, Screen.width, Screen.height);
```

```
            }
        }

        if(cam == null)
        {

gameManager.showErro(gameManager.messages.getErroAbrirCamera(), true,
false);
        }
        else {
            cam.Play();
            cameraImage.texture = cam;
        }
    }
    public byte[] TakePhoto()
    {
        cam.Pause();
        Texture2D photo = new Texture2D(cam.width, cam.height);
        photo.SetPixels(cam.GetPixels());
        photo.Apply();
        byte[] bytes = photo.EncodeToPNG();
        cam.Stop();

        return bytes;
    }

}

using UnityEngine;

public class PlayerController : MonoBehaviour
{
    public AudioClip collectableAudio;

    void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("Collectable"))
        {
```

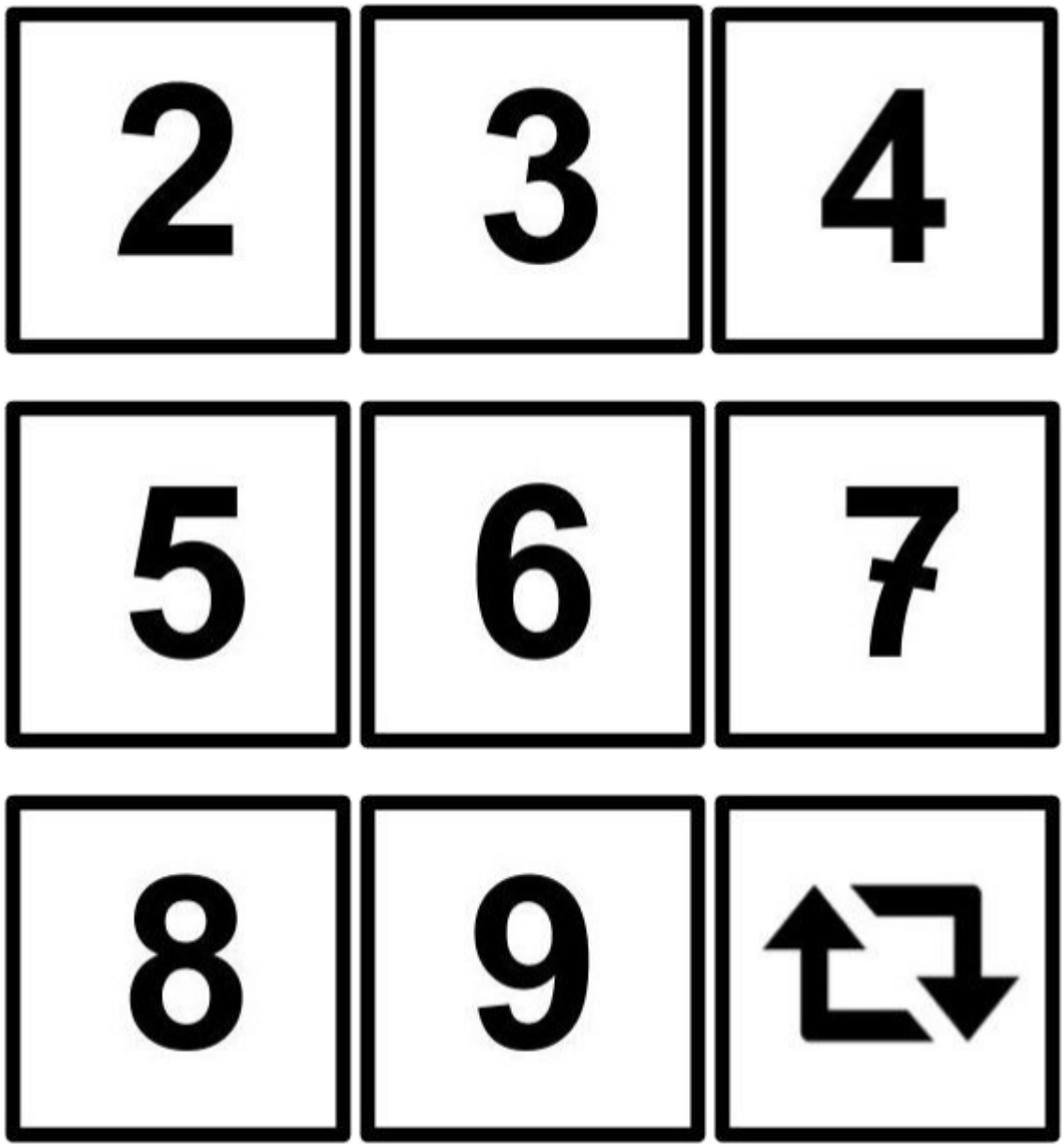
```
        other.gameObject.SetActive(false);
                SoundManager.instance.PlaySingle(collectableAudio);

GameManager.instance.checkEndGameCollectable(other.transform.position);
            }
            else
            {
                StartCoroutine(GameManager.instance.doDefeat(true));
            }
        }
    }

    using UnityEngine;
    using UnityEngine.UI;

    public class SelectLevelManager : MonoBehaviour {

        public Text title;
        public int maxLevel;
        private int page, levelReached;
        public Button nextPage, backPage;
        public GameObject buttonLevel;
        public GameObject panelButtonsLevel;

        void Start () {

            levelReached = PlayerPrefs.GetInt("levelReached", 1);
            page = 0;

            title.text = GameManager.instance.messages.getTituloTelaSelecao();
            backPage.interactable = false;
            nextPageInteractable();
            printPage();
            GameManager.instance.setSelectLevel(this);
        }

        private void printPage()
        {
            GameObject lockObject, button, textButton;
```

```
            int position = 0;

            foreach (Transform child in panelButtonsLevel.transform)
            {
                GameObject.Destroy(child.gameObject);
            }

            for (int i = 1+(page*10); i <= (page+1)*10 && i <= maxLevel; i++)
            {
                button = Instantiate(buttonLevel) as GameObject;
                                                button.GetComponent<RectTransform>
().SetParent(panelButtonsLevel.transform, false);
                if (position < 5) {
                        button.transform.localPosition = getPositionInstance(position, 0,
button.GetComponent<RectTransform>().rect.width,
button.GetComponent<RectTransform>().rect.height);
                }
                else
                {
                    button.transform.localPosition = getPositionInstance(position-5, -1,
button.GetComponent<RectTransform>().rect.width,
button.GetComponent<RectTransform>().rect.height);
                }
                button.GetComponent<Button>().onClick.RemoveAllListeners();
                string temp = i.ToString();
                        button.GetComponent<Button>().onClick.AddListener(delegate {
GameManager.instance.loadLevel(temp);});
                textButton = button.transform.GetChild(0).gameObject;
                textButton.GetComponent<Text>().text = i.ToString();
                lockObject = button.transform.GetChild(1).gameObject;
                if (i > levelReached)
                {
                    button.GetComponent<Button>().interactable = false;
                    lockObject.SetActive(true);
                }
                else
                {
                    button.GetComponent<Button>().interactable = true;
```

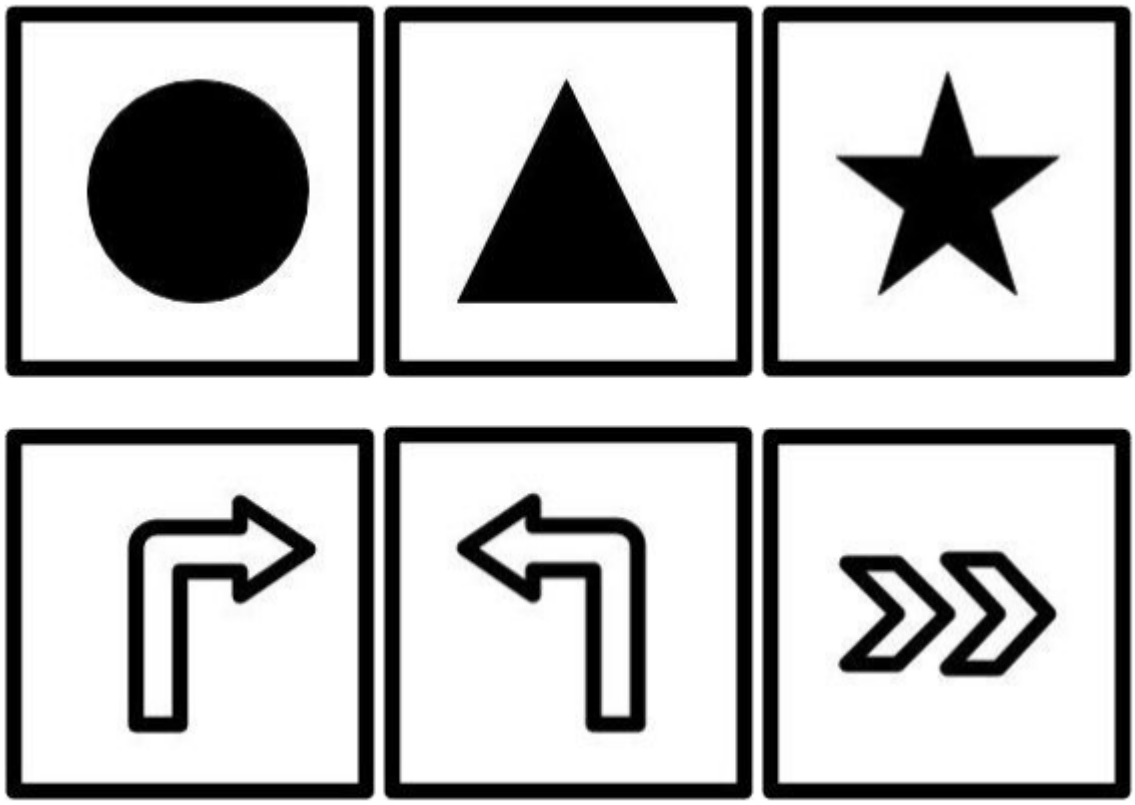
```
        lockObject.SetActive(false);
                }
                position++;
            }
        }

            private Vector3 getPositionInstance(float x, float y, float offsetX, float
offsetY)
        {
            return new Vector3(x * offsetX + -220, y * offsetY + 150, 0f);
        }

        public void nextPageLevelClick()
        {
            page++;
            backPage.interactable = true;
            nextPageInteractable();
            printPage();
        }

        public void nextPageInteractable()
        {
            if(maxLevel > (page+1) * 10)
            {
                nextPage.interactable = true;
            }
            else
            {
                nextPage.interactable = false;
            }
        }

        public void backPageSelectLevelClick()
        {
            page--;
            nextPage.interactable = true;
            if(page == 0)
            {
                backPage.interactable = false;
```

```
        }
        printPage();
    }
}

public class SoundButtonController : UnityEngine.MonoBehaviour
{
 void Start () {
       SoundManager.instance.soundInit(gameObject);
    }
}
```

ANEXO A — PEÇAS LOOP



ANEXO B — PEÇAS DE MOVIMENTAÇÃO



Space Code: Um jogo para o ensino de Pensamento Computacional utilizando Interfaces Tangíveis

Yuri Kayser da Rosa, Everton Schafaschek Coelho

Departamento de Informatica e Estatística
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil
ykr337@gmail.com, evertonscoelho@gmail.com

Abstract. *Since Wing (2006) coined the term, Computational Thinking has been highlighted by many authors (like BLIKSTEIN 2008 and STEPHENSON 2016) as a new and fundamental tool to read, comprehend and solve problems in a increasingly digital world.*

Knowing how important this concept is and with the objective of helping the propagation and learning of it, this work presents the creation of Space Code, an educational game for the Android platform which objective is to serve as a support toll in the teaching of Computational Thinking for children between 7 and 12 years old.

The developed game uses Tangible Interfaces that are manipulated by the users to build algorithms and solve the problems to which they are exposed. The application also uses Computer Vision techniques to identify the tangible objects and perform the proposed solution.

Resumo. *Desde que Wing (2006) cunhou o termo, o Pensamento Computacional vem sendo destacado por diversos autores (como BLIKSTEIN 2008 a STEPHENSON 2011) como uma nova e fundamental ferramenta para ler, compreender e dar soluções a problemas em um mundo cada vez mais digital.*

Atento a importância deste conceito e com o objetivo de ajudar na difusão e aprendizado do mesmo, este trabalho apresenta a criação do Space Code, um jogo educacional na plataforma Android cujo objetivo é servir como uma ferramenta de apoio ao ensino de Pensamento Computacional para crianças entre 7 a 12 anos.

O jogo criado faz uso Interfaces Tangíveis que os usuários manipulam para montar algoritmos e assim resolverem os problemas a que são expostos. A aplicação desenvolvida também utiliza técnicas de Visão Computacional para identificar os objetos tangíveis e performar a solução proposta.

Apendice C – Artigo

1. Introdução

O desenvolvimento de computadores cada vez mais baratos e pequenos trouxe estas máquinas das fábricas e universidades para dentro de nossas casas e salas de aula, possibilitando o acesso a estes dispositivos, principalmente os smartphones, por pessoas das mais variadas idades e classes sociais. Segundo a 28ª Pesquisa Anual de Administração e uso de Tecnologia da Informação nas Empresas, realizada pela Fundação Getúlio Vargas de São Paulo (FGV, 2017) , o número de smartphones no Brasil em Maio de 2017 era de 198 milhões, sendo maior que computadores, 166 milhões, o que resulta em uma densidade de 1,8 dispositivos por habitantes.

Este novo mundo em que vivemos exige um extenso conjunto de habilidades para o pleno exercício da cidadania, como a computação e o Pensamento Computacional, e é justo oferecer às nossas crianças a oportunidade de aprender tais conceitos, para que possam ser consumidores e criadores educados de novas tecnologias que podem melhorar a vida de todos (BLIKSTEIN , 2008, p. 1). e (VON WANGENHEIM; NUNES; DOS SANTOS, 2014).

Entretanto, a forma de ensino atual deste conceitos pode ser vista como problemática, pois percebe-se que o ensino de computação nas escolas está voltado principalmente para o uso das tecnologias e não para a compreensão e aplicação das mesmas. (VON WANGENHEIM; NUNES; DOS SANTOS, 2014).

É neste contexto que a utilização de Interfaces Tangíveis pode ajudar, proporcionando a oportunidade de criar sistemas físicos computacionalmente aumentados. FALCÃO e GOMES (2007) sugere que o uso de Interfaces Tangíveis para a educação traz vantagens como maior engajamento sensorial (uso de mais sentidos para construir o conhecimento), acessibilidade, aprendizagem em grupo e aprendizagem divertida, o que estimula a comunicação, reflexão, imaginação, criatividade em diferentes níveis de abstração e aumenta a consciência sobre a experiência vivenciada.

Tendo esses pontos em vista, foi criado o Space Code, um aplicativo mobile para a plataforma android, que consiste em um jogo para o desenvolvimento de conceitos relativos ao Pensamento Computacional utilizando como método de interação as Interfaces Tangíveis.

2. Conceitos básicos

Pensamento Computacional é um termo cunhado por Wing (2006) para uma forma de pensar e encontrar soluções para os problemas de diversas situações do dia a dia utilizando como base os fundamentos da Ciência da Computação e Matemática. Segundo Wing (2006). Pensamento Computacional não é saber programação e sim pensar como um cientista da computação. A pesquisadora ainda afirma que essa deveria ser uma habilidade básica, assim como ler, escrever, falar e fazer operações aritméticas.

Complementando Wing, BLIKSTEIN (2008) afirma que Pensamento Computacional não é saber realizar as tarefas básicas no computador, como navegar na internet ou enviar um e-mail, mas sim utilizar o computador a fim de incrementar o poder cognitivo e operacional humano de realizar suas tarefas, aumentando a produtividade, inventividade e criatividade.

Quanto às Interfaces Tangíveis podem ser definidas como o uso de objetos físicos, como sendo meios de entrada e/ou saída de uma aplicação, tirando proveito das vantagens da manipulação de objetos reais e das formas de interação providas pela tecnologia.

Entre os vários campos de aplicação das Interfaces Tangíveis, a área de ensino e aprendizado tem sido uma das que mais chama a atenção (MARSHALL, 2007). Isto se deve a uma visão geral do mundo educacional de que atividades práticas ou com uso de objetos manipuláveis facilitam o aprendizado. (MARSHALL, 2007).

MARSHALL (2007) propõe duas razões para as quais TUIs podem ser adequadas a este tipo de atividade. A primeira refere-se à capacidade das TUIs de promover interações mais fluídas e intuitivas e de proporcionarem um rápido feedback, o que facilita o processo de experimentação e teste. A segunda refere-se às teorias de que manipulações físicas de materiais podem ajudar no aprendizado. Assim, sendo as TUIs interfaces extremamente manipuláveis, elas podem proporcionar algum ganho de desempenho.

3. Trabalhos Correlatos

Como trabalhos correlatos foram selecionados jogos voltados especificamente para o ensino de Pensamento Computacional. Os jogos foram separados em três grupos diferentes conforme o tipo de interface utilizada:

- **Jogos de interface puramente gráfica:** São jogos que utilizam apenas a interface gráfica do software como forma de interação com o usuário, não proporcionando uma experiência física e manipulável ao usuário
- **Jogos de interface puramente tangível:** Jogos que utilizam apenas de manipulativos digitais, promovendo uma experiência física interessante mas que geralmente possui custos mais altos, uma vez que conta com equipamentos de computação embarcada, que tendem a ser mais caros.
- **Jogos de interface híbrida:** São jogos que utilizam tanto as interfaces gráficas dos softwares como os objetos manipuláveis das Interfaces Tangíveis, utilizando as vantagens de ambas as interfaces com custos menores.

Após analisar os trabalhos correlatos foi elaborada uma proposta de jogo que tivesse como forma de interação uma interface híbrida, que fosse gratuito e de código aberto, voltado para a plataforma mobile Android, que possui custos menores de aquisição dos aparelhos, e que adotasse mecanismos de internacionalização de software. O fruto desta proposta é o Space Code.

4. Space Code

A ideia do jogo é expor o usuário a um desafio na tela do celular, em que ele deve guiar um foguete pelo tabuleiro, sem esbarrar nas paredes ou obstáculos do caminho, com o objetivo de capturar todas as estrelas existentes no cenário.

Figura 1. Exemplo de desafio

Para que a nave se movimente, é necessário que o usuário utilize o conjunto de peças físicas que simbolizam as instruções aceitas pelo jogo e as organize em uma sequência lógica que .levará o foguete ao seu destino final.

Há dois conjuntos de peças disponíveis. A figura 2 mostra as instruções que simbolizam funções e movimentação.

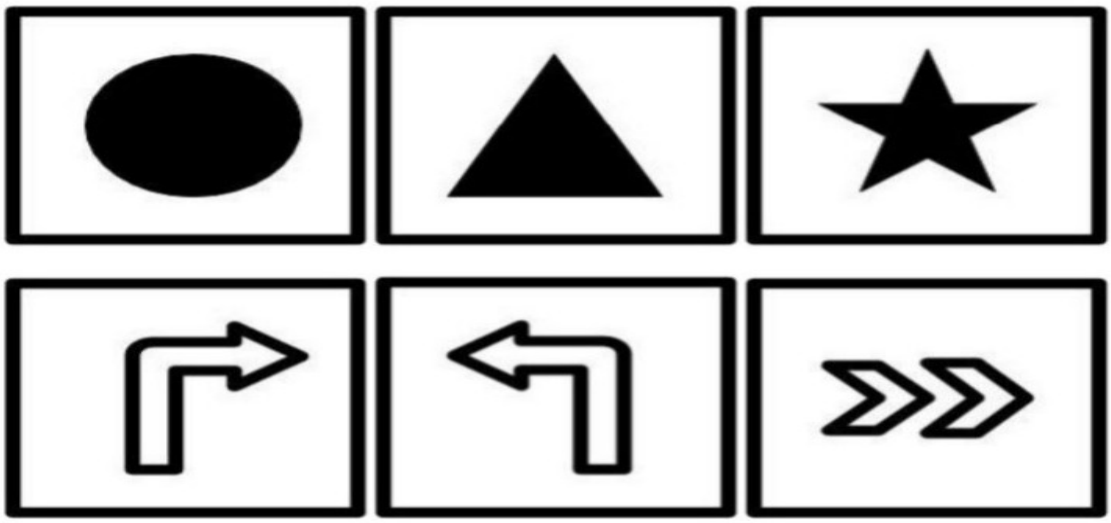


Figura 2. Peças de função e movimentação

A figura 3 mostra o conjunto de instruções que pode ser utilizado para montar estruturas de repetição

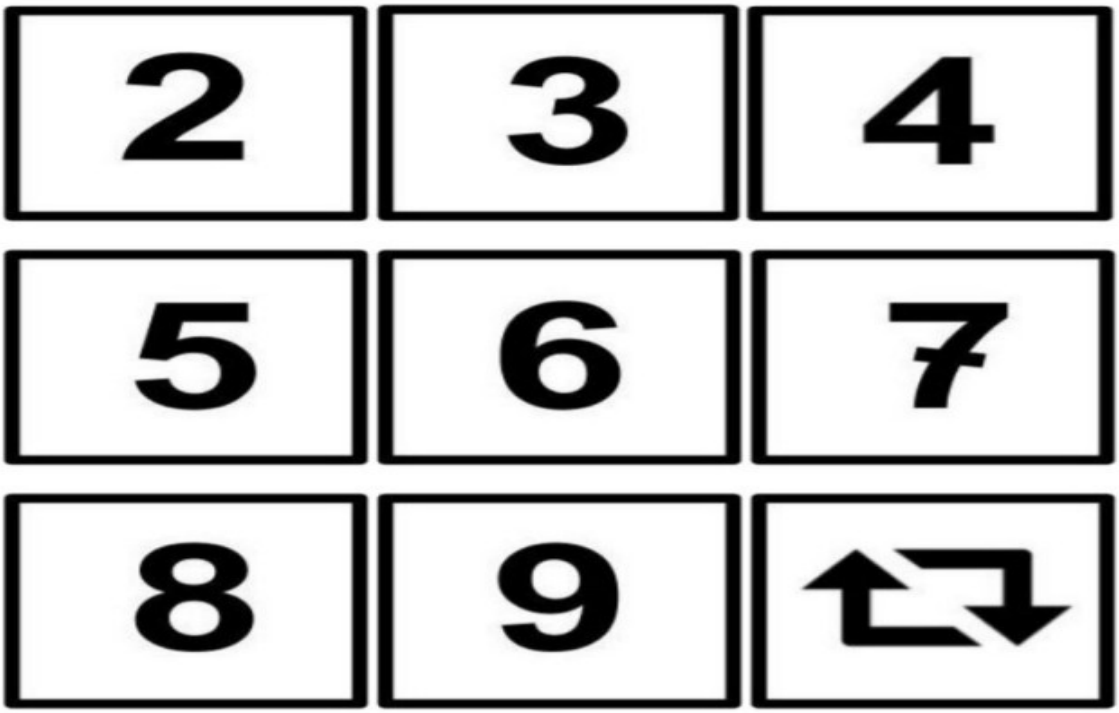


Figura 3. Peças de repetição

Após organizar as peças, o usuário deve utilizar a câmera do dispositivo para capturar uma imagem das mesmas. Esta foto será então processada para que sejam identificadas as peças utilizadas pelo jogador. Após a identificação, o aplicativo se encarrega de realizar as instruções desejadas, mostrando à criança a movimentação do foguete pelo tabuleiro de acordo com o algoritmo por ela descrito.

4.1 Estrutura Básica

A aplicação foi dividida em dois módulos: o aplicativo, que contém o jogo, e um Webservice que abriga o módulo de Visão Computacional responsável por identificar as peças na imagem. A imagem 4 mostra a estrutura e o fluxo de comunicação da aplicação.

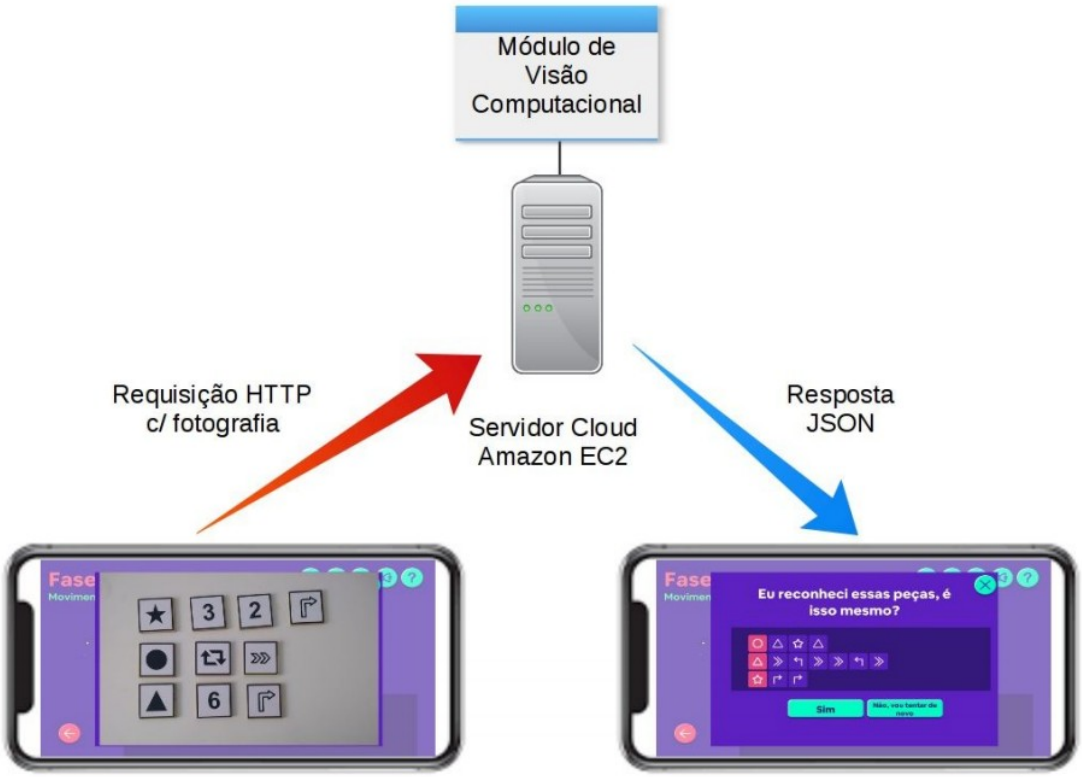


Figure 4. Estrutura da aplicação

O aplicativo foi desenvolvido utilizando a Game Engine Unity e é o responsável por mostrar ao usuário o desafio a ser cumprido e por dar instruções, através de janelas de ajuda, sobre como o jogo funciona e como as peças podem ser utilizadas. As fases do jgo podem ser montadas através de um arquivo no formato JSON em que é possível especificar os elementos que compõem a fase, como o foguete, as estrelas, paredes e obstáculos, e a posição dos mesmos na tela.

O módulo de Visão Computacional foi feito utilizando a biblioteca open source OpenCV na linguagem Python e hospedado em uma máquina do tipo EC2 na Amazon. O servidor recebe uma requisição http enviada pelo aplicativo e fica responsável por processar a imagem com o objetivo de reconhecer as instruções dadas pelo usuário. Para isso, a foto passa primeiro por algoritmos que removem ruídos e transformam as cores da foto em preto e branco. Na imagem resultante o código procura pelos contornos das peças, os ordena, e compara cada um deles com imagens base das instruções reconhecidas pelo sistema, verificando com qual peça o contorno mais se assemelha ou se não possui semelhança alguma. O resultado é então enviado novamente para o aplicativo que fica responsável por mostrar, na interface gráfica, as instruções identificadas e por executá-las.

Durante a fase de desenvolvimento preocupou-se em fazer com que os dois módulos fossem desacoplados, permitindo que alterações em um não influenciassem o funcionamento do outro.

5. Resultados

Ao final do projeto foi gerado um aplicativo executável e dispositivos da plataforma Android e publicado na Google Play Store (SpaceCode, 2018) e o código foi disponibilizado em um repositório público no site Github (Coelho,2018).

O jogo possui algumas limitações, como por exemplo o fato de que o dispositivo mobile precisa estar conectado à internet para que possa se comunicar com o servidor que contém o módulo de Visão Computacional. Uma outra limitação é que o usuário deve manter o dispositivo em posição estática ao capturar a foto, pelo menos até que a mensagem “Carregando” apareça na tela, para conseguir resultados dos comandos mais precisos. Também é necessário que o aparelho possua uma câmera fotográfica traseira funcional.

Foram realizados também alguns testes sobre o módulo de Visão Computacional a fim de garantir sua eficiência e identificar os melhores cenários e limites do sistema. No geral, o módulo se saiu bem ao identificar as peças nas imagens, tendo como cenário ideal uma foto tirada em paralelo às peças, em um ambiente bem iluminado, com pouca ou nenhuma sombra incidindo sobre as peças e com as mesmas dispostas em um fundo branco, como pode ser visto na figura 5.

Figure 4. Estrutura da aplicação

Como limitação, há problemas em reconhecer as peças em imagens onde há uma grande incidência de sombras ou em que há reflexos de luz sobre os objetos. Peças que não estão com as bordas bem definidas, seja por desgaste na pintura ou por estas partes terem ficado fora da imagem, ou em que as mesmas estão danificadas, não são identificadas.

O código referente ao módulo de Visão Computacional e ao conjunto de testes também está disponível em um repositório do Github (Coelho, 2018).

6. Conclusões e trabalhos futuros

A fase de desenvolvimento foi repleta de desafios. Tanto a produção do aplicativo utilizando o Unity quanto o desenvolvimento dos algoritmos de Visão Computacional utilizando o OpenCV exigiram uma grande curva de aprendizado para que houvesse sucesso na implementação, mas ambas supriram completamente as necessidades do projeto. Escolher ferramentas bastante utilizadas foi de fundamental importância, pois em diversos momentos foi necessário recorrer a fóruns de desenvolvimento para esclarecer dúvidas ou buscar ideias.

Mesmo diante das dificuldades, todos os artefatos de software propostos foram completamente desenvolvidos, encontram-se em repositórios públicos, e o jogo em sí pode ser utilizado por quem desejar, uma vez que se encontra publicado na loja de aplicativos da plataforma Android.

A finalização deste projeto abre portas para diversas melhorias e trabalhos futuros. No aplicativo podem ser implementadas novas traduções para outros idiomas, ampliando a questão da internacionalização do software.

No módulo de Visão Computacional é possível que a utilização de métodos de Deep Learning, bem como melhorias no processamento de imagem, principalmente no tratamento de sombras, possa gerar melhorias significativas no reconhecimento correto das instruções.

Também é possível tentar transferir este módulo para dentro do dispositivo mobile, evitando assim que o aplicativo fique dependente de conexão com a internet para funcionar. Novos tipos de instruções podem ser inseridos no sistema, ampliando a variedade de desafios possíveis e aumentando a gama de conceitos abordados nas fases.

A especificação das fases em arquivos no formato JSON possibilita que novos níveis sejam facilmente criados, o que abre espaço para diversas evoluções deste trabalho. É possível explorar essa funcionalidade para criar fases específicas para algum curso ou aula, criar fases com outras dificuldades abordando novos conceitos e problemáticas, aumentando a faixa etária alvo do jogo, e permitir que os próprios jogadores criem as suas fases (aumentando assim o engajamento e estimulando a criatividade) e desafiem outros usuários.

Por fim, devido à limitações de tempo e de escopo deste trabalho, não foi possível fazer uma análise da efetividade do uso desta aplicação como ferramenta de apoio ao ensino de Pensamento Computacional. Esta análise poderá ser feita futuramente, em um projeto piloto, definindo um conteúdo programático a ser abordado tendo o aplicativo como auxílio e analisando os resultados obtidos.

References

Barr, Valerie; Stephenson, Chris. (2011) “Bringing Computational Thinking to K-12: what is involved and what is the role of the computer science education community?”, Acm Inroads, v. 2, n. 1, p. 48-54, 2011.

Blikstein, Paulo. (2008) “O Pensamento Computacional e a reinvenção do computador na educação”, http://www.blikstein.com/paulo/documents/online/ol_pensamento_computacional.html Acesso em:10 de Novembro de 2017 .

FGV, (2017) “28 Pesquisa Anual do uso de TI 2017 ”, https://eaesp.fgv.br/ensinoeconhecimento/centros/cia/pesquisa, Novembro de 2017.

Von Wangenhein, Cristiane G. e Nunes, Vinicius R. (2014) “Ensino de computação com Scratch no ensino fundamental – um estudo de caso”, Revista Brasileira de informatica na educação, v. 22, n. 03, p. 115.

Wing, Jeannette M. (2006) “Computational Thinking”, Communications of the ACM. Nova York – NY, v. 49, n. 3, p. 33-35.

Falcão, Taciana Pontual e Gomes, Alex Sandro. (2007) “Interfaces tangíveis para a educação”, In: BRAZILLIAN SIMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMATICA NA EDUCAÇÃO – SBIE) p. 579-589.

Marshall, Paul (2007) “Do Tangible Interfaces enhance Learning?”, In: PROCEEDINGS OF THE 1ST INTERNATIONAL CONFERENCE ON TANGIBLE AND EMBEDED INTERACTION.

Falcão, Taciana Pontual e Gomes, Alex Sandro. (2007) “Interfaces tangíveis para a educação”, In: BRAZILLIAN SIMPOSIUM ON COMPUTERS IN EDUCATION (SIMPÓSIO BRASILEIRO DE INFORMATICA NA EDUCAÇÃO – SBIE) p. 579-589.

COELHO, Everton S.. Repositórios do usuário evertonscoelho. 2018. Disponível em: <https://github.com/evertonscoelho?tab=repositories>. Acesso em: 23 nov. 2018.

SPACE CODE. Space Code. 2018. Disponível em: <https://play.google.com/store/apps/details?id=com.EvertonYuri.TCC>. Acesso em: 5 dez. 2018.